

TSX-Plus
Reference Manual



ssh computer systems, inc.

1875
1875



1875

TSX-Plus
Reference Manual

Fourth Edition
First Printing -- September, 1984

Copyright © 1980, 1981, 1982, 1983, 1984.
S&H Computer Systems, Inc.
1027 17th Avenue South
Nashville, Tennessee USA
37212-2299
(615)-327-3670

The information in this document is subject to change without notice and should not be construed as a commitment by S & H Computer Systems Inc. S & H assumes no responsibility for any errors that may appear in this document.

NOTE: TSX, TSX-Plus, PRO/TSX-Plus, COBOL-Plus, PRO/COBOL-Plus, SORT-Plus and RTSORT are proprietary products owned and developed by S&H Computer Systems, Inc., Nashville, Tennessee, USA. The use of these products is governed by a licensing agreement that prohibits the licensing or distribution of these products except by authorized dealers. Unless otherwise noted in the licensing agreement, each copy of these products may be used only with a single computer at a single site. S&H will seek legal redress for any unauthorized use of these products.

Questions regarding the licensing arrangements for these products should be addressed to S&H Computer Systems, Inc., 1027 17th Ave. South, Nashville, Tennessee 37212-2299, (615)-327-3670, TELEX 786577 S AND H UD.

TSX, TSX-Plus, PRO/TSX-Plus, COBOL-Plus, PRO/COBOL-Plus, SORT-Plus and RTSORT are trademarks of S&H Computer Systems, Inc. DEC, DIBOL, PDP, Professional, Q-BUS, RT-11, UNIBUS, VAX, VMS and VT are trademarks of Digital Equipment Corporation. DBL is a trademark of Digital Information Systems Corporation.

CONTENTS

INTRODUCTION	1
Management of system resources	2
Summary of chapter contents	4

Chapter 1

BASIC OPERATION	9
Logging on	9
Logging off	10
Control characters	11
Single line editor	12

Chapter 2

KEYBOARD COMMANDS	
Keyboard command interpretation	17
User defined commands	20
User Command Interface	24
Keyboard commands	28
ACCESS Command	28
ALLOCATE Command	28
ASSIGN Command	29
BACKUP Command	30
BOOT Command	31
BYE Command	31
COBOL Command	31
COMPILE Command	32
COPY Command	32
CREATE Command	32
DATE Command	32
DEALLOCATE Command	32
DEASSIGN Command	33
DELETE Command	33
DETACH Command	33
DIBOL Command	35
DIFFERENCES Command	35
DIRECTORY Command	35
DISMOUNT Command	35
DISPLAY Command	36
DUMP Command	36
EDIT Command	36
EXECUTE Command	37
FORM Command	37
FORTRAN Command	37
HELP Command	37
INITIALIZE Command	37
KILL Command	38
KJOB Command	39
LIBRARY Command	39

LINK Command	39
LOGOFF Command	39
MACRO Command	39
MAKE Command	39
MEMORY Command	40
MONITOR Command	41
MOUNT Command	41
MUNG Command	43
OFF Command	43
OPERATOR Command	43
PAUSE Command	44
PRINT Command	44
PROTECT Command	44
R Command	44
RENAME Command	47
RESET Command	47
RUN Command	47
SEND Command	48
SET Command	48
SET CACHE	49
SET CCL	49
SET CL	50
BININ	52
BINOUT	52
CR	52
CTRL	52
DTR	52
FORM	52
FORMO	53
LC	53
LENGTH	53
LFIN	53
LFOUT	53
LINE	53
SKIP	54
SPEED	54
TAB	54
TOP	54
WIDTH	54
SET CORTIM	54
SET CTRLD	55
SET EDIT	55
SET EMT	55
SET ERROR	56
SET HIPRCT	56
SET IND	57
SET INTIOC	57
SET IO	57
SET KMON	57
SET LANGUAGE	59
SET LD	59
SET LOG	59
SET LOGOFF	60
SET MAXPRIORITY	60

SET NUMDC	61
SET PRIORITY	61
SET PROMPT	62
SET QUANxx	62
SET SIGNAL	63
SET SL	63
ASK	63
K52	63
KED	64
KEX	64
LEARN	64
LET	64
OFF	64
ON	64
RT11	64
SYSGEN	64
TTYIN	64
VT52	64
VT62	64
VT100	64
VT101	64
VT102	64
VT200	65
WIDTH	65
SET TERMINAL	65
SET TT	65
ADM3A	66
AUTOBAUD	66
DEAD	66
DECWRITER	66
DEFER	66
DIABLO	67
ECHO	67
EIGHTBIT	67
FORM	67
FORMO	67
GAG	67
HAZELTINE	67
LA36	67
LA120	67
LC	68
PHONE	68
PRIVILEGE	68
QUIET	68
QUME	68
SCOPE	68
SINGLE	69
SPEED	69
TAB	69
TAPE	69
VT50	69
VT52	69
VT100	69
VT200	70

WAIT	70
SET UCL	70
SET VM	71
SET WILDCARDS	72
SHOW Command	72
SHOW ALL	72
SHOW ALLOCATE	73
SHOW ASSIGNS	73
SHOW CACHE	73
SHOW CL	73
SHOW COMMANDS	74
SHOW CONFIGURATION	74
SHOW CORTIM	75
SHOW DEVICES	75
SHOW HIPRCT	76
SHOW INTIOC	76
SHOW JOBS	76
SHOW MEMORY	76
SHOW MOUNTS	77
SHOW NUMDC	77
SHOW PRIORITY	78
SHOW QUANxx	78
SHOW QUEUE	78
SHOW RUN-TIMES	79
SHOW SPOOL	79
SHOW SUBSETS	79
SHOW TERMINALS	79
SHOW USE	81
SPOOL Command	81
SQUEEZE Command	82
SYSTAT Command	83
TECO Command	84
TIME Command	84
TYPE Command	84
UCL Command	84
UNPROTECT Command	85
USE Command	85
WHO Command	85
\$STOP Command	85
\$SHUTDOWN Command	85
RT-11 Commands not supported by TSX-Plus	86

Chapter 3

COMMAND FILES	87
Invoking command files	88
Parameter strings	88
Comments in command files	89
Command file control characters	90
PAUSE Command	91
DISPLAY Command	91

Chapter 4

VIRTUAL TIME-SHARING LINES AND DETACHED JOBS	93
Virtual lines	93
Detached jobs	94
The DETACH Command	95
Starting a detached job	95
Checking detached job status	96
Aborting a detached job	97
Detached job control EMTs	97
Starting a detached job	97
Aborting a detached job	99
Checking detached job status	100

Chapter 5

DEVICE SPOOLING	101
The concept of device spooling	101
Directing output to spooled devices	101
Operation of the spooler	102
The SPOOL Command	102
FORM and LOCK functions	103
ALIGN function	103
DELETE function	104
SKIP function	104
BACK function	105
STAT function	105
SING and MULT functions	105
HOLD and NOHOLD functions	106
Use of special forms with spooled devices	108
Form alignment procedure	109

Chapter 6

PROGRAM CONTROLLED TERMINAL OPTIONS	111
Terminal input/output handling	111
Program controlled terminal options	113
Set rubout filler character	116
Set VT52, VT100 and VT200 escape-letter activation . .	116
Define new activation character	117
Control character echoing	117
Disable virtual line use	117
Control lower case input	117
Control character echoing	117
Set transparency mode of output	118
Control command file input	118
Reset activation character	118
Set activation on field width	118
Turn on high-efficiency mode	118
Turn on single-character activation mode	119
Turn off single-character activation mode	119
Enable non-wait TT input testing	119

Set field width limit	120
Control tape mode	120
Control line-feed echo following carriage-return . . .	120

Chapter 7

TSX-Plus EMT'S	121
Determining if a job is under TSX-Plus	121
Determining the TSX-Plus line number	122
Determining the terminal type	123
Determining or changing the user name	124
Controlling the size of a job	125
Obtaining TSX-Plus system values (.GVAL)	126
Determining job status information	128
Job monitoring	131
Setting job priority	136
Forcing [non]interactive job characteristics	138
Sending a message to another line	140
Mount a file structure	140
Dismount a file structure	142
Set terminal read time-out value	143
Establish break sentinel control	144
Checking for terminal input errors	145
Checking for activation characters	147
Printing a block of characters	148
Accepting a block of characters	149
Program controlled terminal options	150
Setting terminal baud rates	151
Assigning a CL unit to a time-sharing line	152
Allocating a device for exclusive use	156
Controlling high-efficiency terminal mode	157
Determining number of free spool blocks	159
Set/Reset ODT activation mode	159
Determining file directory information	161
Setting file creation time	163

Chapter 8

TSX-Plus JOB ENVIRONMENT	165
Virtual and physical memory	165
User virtual address mapping	166
Normal programs and virtual programs	167
Extended memory regions	167
Shared run-time systems	168
Access to system I/O page	169
VM memory based pseudo-disk	169

Chapter 9

SHARED FILE RECORD LOCKING	171
Opening a shared file	171
Saving the status of a shared file channel	175
Waiting for a locked block	178
Trying to lock a block	180
Unlocking a specific block	182
Unlocking all locked blocks in a file	183
Checking for writes to a shared file	183
Data caching	184

Chapter 10

MESSAGE COMMUNICATION FACILITIES	187
Message channels	187
Sending a message	187
Checking for pending messages	189
Waiting for a message	190
Scheduling a message completion routine	192

Chapter 11

REAL-TIME PROGRAM SUPPORT	197
Accessing the I/O page	198
EMT to map the I/O page into the program	198
EMT to remap the program region to RMON	200
EMT to peek at the I/O page	201
EMT to poke into the I/O page	203
EMT to bit-set a value into the I/O page	203
EMT to bit-clear a value into the I/O page	206
Mapping to a physical memory region	206
Requesting exclusive system control	209
Locking a job in memory	210
Unlocking a job from memory	211
Suspending/Resuming program execution	212
Converting a virtual address to a physical address	212
Specifying a program-abort device reset list	214
Setting processor priority level	214
Setting job execution priority	215
Connecting interrupts to real-time jobs	216
Interrupt service routines	219
Interrupt completion routines	224
Releasing an interrupt connection	228
Scheduling a completion routine	229
Adapting real-time programs to TSX-Plus	230

Chapter 12

SHARED RUN-TIME SYSTEM SUPPORT	233
Associating a run-time system with a job	233
Mapping a run-time system into a job's region	235

Chapter 13

PERFORMANCE MONITOR FEATURE	237
Starting a performance analysis	237
Displaying the results of the analysis	238
Performance monitor control EMT's	239
Initializing a performance analysis	239
Starting a performance analysis	243
Stopping a performance analysis	243
Terminating a performance analysis	243

Chapter 14

TSX-Plus RESTRICTIONS	245
System service call (EMT) differences	245
Programs not supported by TSX-Plus	247
Special program suggestions	247

Appendix A

SETSIZ PROGRAM	251
Running the SETSIZ program	252
Setting total allocation for a SAV file	253
Setting amount of dynamic memory space	253
Setting virtual-image flag in SAV file	253

Appendix B

DIBOL TSX-PLUS SUPPORT SUBROUTINES	255
Record locking subroutines	255
Opening a shared file	255
Locking and reading a record	256
Writing a record	257
Unlocking records	258
Closing a shared file	258
Record locking example	258
Modifying programs for TSX-Plus	258

Message communication subroutines	258
Message channels	258
Sending a message	259
Checking for pending messages	260
Waiting for a message	260
Message examples	261
Using the subroutines	261
Miscellaneous functions	261
Determining the TSX-Plus line number	261

Appendix C

FILTIM PROGRAM	263
--------------------------	-----

Appendix D

RT-11 & TSX-Plus EMT CODES	265
RT-11 EMT codes	265
TSX-Plus EMT codes	267

Appendix E

SUBROUTINES USED IN EXAMPLE PROGRAMS	269
PRT OCT - Print an octal value	269
PRT DEC - Print a decimal value	269
PRT DE2 - Print a 2 digit decimal value	270
PRT R50 - Print a RAD50 word at the terminal	271
R50 ASC - Convert a RAD50 string into an ASCII string	271
DSP DAT - Print a date value at the terminal	272
DSPT I3 - Display a 3-second format time value	274
ACRT I3 - Convert a time value to special 3-second format	274

Appendix F

TSX-Plus USER ERROR MESSAGES	277
--	-----

Appendix G

LOGICAL SUBSET DISKS	289
--------------------------------	-----

Appendix H

JOB EXECUTION PRIORITIES	291
------------------------------------	-----

Appendix I

PROGRAM DEBUGGER	293
Debugger requirements	293
Invoking the debugger	294
RUN/DEBUG Switch	294
Control-D break	294
Commands	295
Control-D breakpoints	297
Address relocation	297
Internal registers	298
Data watchpoints	298
Symbolic instruction decoding	299
Special notes	300

INTRODUCTION

TSX-Plus is a high-performance operating system for Digital Equipment Corporation PDP-11 and LSI-11 computers supporting up to 31 concurrent time-sharing users. TSX-Plus provides a multi-user programming environment that is similar to extended-memory (XM) RT-11.

1. TSX-Plus keyboard commands are compatible with those of RT-11.
2. TSX-Plus supports most RT-11 system service calls (EMTs).
3. Most programs that run under RT-11 will run without modification under TSX-Plus. This includes RT-11 utility programs such as PIP, DUP, DIR, LINK, and MACRO.
4. TSX-Plus uses RT-11 XM version device handlers.
5. TSX-Plus provides PLAS extended memory services such as virtual overlays and virtual arrays, as well as support for extended memory regions.

TSX-Plus can simultaneously support a wide variety of jobs and programming languages including COBOL-Plus, FORTRAN, BASIC, DIBOL, DBL, Pascal, C, MACRO, IND, TECO and KED. TSX-Plus is used in educational, business, scientific and industrial environments. It can concurrently support commercial users doing transaction processing, engineering users performing scientific processing, system programmers doing program development, and real-time process control. Numerous application software packages compatible with TSX-Plus are available from other vendors.

TSX-Plus supports RT-11 system service calls (EMTs) as its basic mode of operation. The result is low system overhead and substantially improved performance over systems that emulate RT-11 services. TSX-Plus overlaps terminal interaction time, I/O wait time, and CPU execution time for all jobs on the system. The result is a tremendous increase in the productivity of the computer system.

In addition to the basic RT-11 functionality, TSX-Plus provides extended features such as: shared file record locking; inter-job message communication; program performance monitoring; command file parameters; logon and usage accounting; directory and data caching; multitasking; and system I/O buffering.

This manual describes all the features unique to TSX-Plus as well as any differences from RT-11. Many of the special features of TSX-Plus are available as EMTs available to the MACRO programmer. Access to these features from other languages requires the appropriate subroutine interfacing.

TSX-Plus will run on any PDP-11 or LSI-11 computer with memory management hardware and at least 128Kb of memory. The system must also have a disk suitable for program swapping (the swapping disk can be used for regular file storage as well). Time-sharing lines and serial printers may be connected to the system through DH(V)-11, DL(V)-11 or DZ(V)-11 communication devices. Both hard-wired and dial-up time-sharing lines are supported by TSX-Plus.

Introduction

MANAGEMENT OF SYSTEM RESOURCES

Memory Management

TSX-Plus uses the memory management facilities of PDP-11 computers to keep several user jobs in memory simultaneously and switch rapidly among them. TSX-Plus protects the system by preventing user jobs from halting the machine or storing outside their program regions. TSX-Plus provides several ways to control the amount of memory used by individual jobs. Programs may be allowed to use up to 64Kb of memory and, if additional space is needed, may also use extended memory regions, virtual overlays and virtual arrays. The system manager may enable job-swapping to accommodate more user jobs than can fit in existing memory.

Execution Scheduling

TSX-Plus provides fast response to interactive jobs but minimizes job-swapping by use of an efficient job-scheduling algorithm. TSX-Plus permits job scheduling on both an absolute priority basis and by a method based on job states. For most applications, the method based on job states is preferred. The state-driven method provides the most transparent time-sharing scheduling, suitable for interactive environments. The absolute priority method always runs the highest priority executable job, when not servicing interrupts, regardless of that job's state. This state-free method is most suitable for an environment in which several real-time jobs must be assigned absolute priorities. TSX-Plus permits both kinds of jobs to co-exist in the same system, with interactive jobs being scheduled whenever higher priority state-free jobs are not executing.

Job priorities may be assigned over a range of 0 to 127. The lowest priority jobs, typically 0 to 19, are reserved for fixed priority jobs which can soak up system idle time without disturbing interactive or real-time jobs. The medium priority range, typically 20 to 79, is assigned to interactive jobs which are scheduled according to a unique and efficient algorithm which makes time-sharing nearly transparent to several users. The highest priority range, typically 80 to 127, is reserved for jobs which must execute according to a rigid priority scheme such as might be found in a real-time environment. In addition, real-time jobs may execute interrupt service routines at fork level processing or schedule interrupt completion routines to run as fixed-high-priority jobs.

Job scheduling is controlled by several system parameters relating job priorities, system timing and other events. The TSX-Plus System Manager's Guide includes a more complete description of job priority and scheduling.

Directory and Data Caching

TSX-Plus provides a mechanism to speed up directory operations by caching device directories. This reduces disk I/O necessary to open existing files. Caching of file data is also possible to further improve system throughput. The information kept in the cache buffers is managed according to a least-recently-used algorithm. Directory and data caching are discussed in the System Manager's Guide.

System Administrative Control

TSX-Plus allows the system manager to limit access to the system through a logon facility and to restrict user access to peripheral devices. These features are described in the TSX-Plus System Manager's Guide.

Time-sharing lines and CL units

TSX-Plus supports time-sharing terminal lines interfaced through several types of interface cards: DH(V)11, DL(V)11 and DZ(V)11. Using the CL feature provided with TSX-Plus, other serial devices such as printers, plotters and communications devices can also be attached using the same types of interfaces. Up to 8 CL units may be attached to TSX-Plus. These devices may either be permanently assigned as CL units or may be used at some times as time-sharing lines and other times as I/O devices by having a CL unit "take over" an inactive time-sharing line. See the SET CL and SHOW CL commands for more information on using CL units. See the TSX-Plus System Manager's Guide for more information on including dedicated and extra CL units during system generation.

Introduction

SUMMARY OF CHAPTER CONTENTS

CHAPTER 1. Basic Operations and Starting Time-sharing Sessions

Chapter 1 of this manual describes the procedure for starting and stopping a time-sharing session with TSX-Plus; the special functions of some control-characters; and use of the single line editor for correction of typing errors in command lines or data entry fields.

CHAPTER 2. Keyboard Commands

TSX-Plus responds to keyboard commands that are simple and natural (PRINT, RUN, COMPILE, TYPE, etc.). TSX-Plus also provides user-definable commands, and includes support for a menu driven command interface. Chapter 2 discusses command interpretation sequence, user-defined commands, the menu interface and lists all TSX-Plus keyboard commands. It provides full descriptions of commands unique to TSX-Plus, describes differences between TSX-Plus and RT-11 commands, and lists the RT-11 commands which are not supported by TSX-Plus. Many of the commands are identical to their RT-11 counterparts and the RT-11 System User's Guide should be consulted for detailed descriptions.

CHAPTER 3. Command Files

Frequently used sets of commands may be stored in a disk file and executed by invoking the command file. Parameters may be passed to the command file and interpreted as though they were included at designated places in the file. This provides an alternative way for users to create their own commands. Command files are described in Chapter 3.

CHAPTER 4. Virtual Lines and Detached Jobs

TSX-Plus provides a facility known as "virtual lines" that allows one time-sharing user to simultaneously control several programs from a single terminal. The user may logically switch among the primary and virtual lines at any time. When a program that is not currently connected to a time-sharing line writes output to the terminal, the output is stored in a system buffer. When the buffer is filled, the program is suspended until the user reattaches to the program and accepts the queued output. Virtual lines are useful in situations in which it is desirable to run a long "number crunching" job without tying up a terminal. Detached jobs are similar, but are not associated with any terminal. Virtual lines and detached jobs are discussed in Chapter 4.

CHAPTER 5. Printer Spooling System

TSX-Plus provides a convenient and powerful facility for automatic spooling of output to line printers and other devices. The spooler may simultaneously drive several devices. If the line printer is spooled, then simply directing output to device "LP" from a program causes the output to be spooled. A spooled file may designate the name of a form on which it is to be printed. When a form change is required, the spooled device is suspended and a message is sent to the operator requesting the form. After mounting the new form, the operator may print a form alignment file. Once a form is mounted, all files requiring the form are printed without further operator intervention. The operator may also lock a particular form on the printer, preventing automatic form change requests. Keyboard commands are provided to check the status of

spooled devices, delete current or pending files from the output queue, and reprint the most recent portion of the current file. Files sent to a spooled device may be released to the printer as data becomes available or held until the output file is closed. The spooling system is discussed in Chapter 5.

CHAPTER 6. Program Controlled Terminal Options

TSX-Plus allows the programmer to modify terminal handling characteristics during program execution. For example, a program may disable character echoing, use single character activation, use high efficiency output mode, enable lower case input, activate on field width, or disable automatic echoing of line-feed after carriage-return. These terminal options may be selected either by issuing the appropriate EMT request or by writing a special sequence of two or three characters to the terminal. Chapter 6 discusses the use of terminal options during program execution.

CHAPTER 7. TSX-Plus EMTs

TSX-Plus supports most of the system service calls (EMTs) provided by RT-11 and, in addition, provides many more to utilize the special features of TSX-Plus. For example, EMTs are provided to determine the TSX-Plus line number and the user name, to send messages to another time-sharing line, to check for terminal input errors, and to check for activation characters. EMTs related to specific features, such as detached jobs, inter-program messages or real-time programming, are described in the relevant chapters. The TSX-Plus EMTs which are not closely related to features described elsewhere are discussed in Chapter 7.

CHAPTER 8. TSX-Plus Job Environment

TSX-Plus supports the use of extended memory regions through EMT calls compatible with those provided by the RT-11 XM monitor. This allows the use of virtual overlays and FORTRAN virtual arrays. Users can also expand programs to use the full 16-bit virtual address space. That is, by giving up access to the I/O page and direct access to fixed offsets in RMON, a program may directly address a full 64Kb. User control of extended memory and virtual job space is discussed in Chapter 8.

CHAPTER 9. Shared Files, Record Locking and Data Caching

TSX-Plus provides a file sharing mechanism whereby several cooperating programs may coordinate their access to common data files. Programs may request different levels of shared file access, and control shared access on a record-by-record basis. Two methods of data caching are also provided: 1) generalized data caching which is enabled when devices are MOUNTed; and 2) record caching which is only available to shared files. Directory caching is also enabled by the MOUNT request. This accelerates directory searching for file LOOKUPs. The use of shared files and data caching is described in Chapter 9.

Introduction

CHAPTER 10. Inter-program Message Communication

TSX-Plus offers a message communication facility that allows running programs to exchange messages. Messages are transmitted through named "message channels". A program can queue messages on one or more message channels. Receiving programs can test for the presence of messages on a named channel and can suspend their execution until a message arrives. Receiving programs may also schedule a completion routine to be entered when a message arrives and continue other processing in the meanwhile. A message can be queued for a program that will run at a later time. The message facility is discussed in Chapter 10

CHAPTER 11. Real-time Support

TSX-Plus provides real-time program support services that allow multiple real-time programs to run concurrently with normal time-sharing operations. Real-time programs may optionally lock themselves in memory, directly access the I/O page, redefine their memory mapping, and connect device interrupts to subroutines within the program. Real-time support is discussed in Chapter 11.

CHAPTER 12. Shared Run-time System Support

TSX-Plus allows one or more shared run-time systems to be mapped into the address space of multiple TSX-Plus time-sharing jobs. This saves memory space when multiple users are running the same types of programs (e.g., COBOL-Plus or DBL) and can also be used in situations where programs wish to communicate through a shared data region. Shared run-time systems are discussed in Chapter 12.

CHAPTER 13. Program Performance Analysis Facility

TSX-Plus includes a performance analysis facility that can be used to monitor the execution of a program and determine what percentage of the run time is spent within certain program regions. When the performance analysis facility is being used, TSX-Plus examines the program being monitored at each clock tick (50 or 60 times per second) and notes the value of the program counter. On completion of the analysis, the TSX-Plus performance reporting program can produce a histogram of the time spent in various parts of the monitored program. Performance analysis is discussed in Chapter 13.

CHAPTER 14. Differences from RT-11

Some inevitable differences exist between RT-11 and TSX-Plus. Chapter 2 describes the additional keyboard commands provided by TSX-Plus, the minor differences in some commands, and the RT-11 keyboard commands not supported by TSX-Plus. Some other differences between RT-11 and TSX-Plus may not be obvious. The FORMAT utility is not supported. A few system service calls (EMTs) behave slightly differently in the two systems and some RT-11 EMTs are not supported by TSX-Plus (notably those supporting multi-terminal operations). These differences are detailed in Chapter 14.

APPENDICES

Appendix A describes the SETSIZ utility program which may be used to control the amount of memory available to programs. Appendix B describes a library of subroutines which are available to the DIBOL user to take advantage of some of the special features of TSX-Plus. Appendix C describes the FILTIM utility which displays directory information about files, including the file creation time. Appendix D provides a table of EMT function and subfunction codes, and brief descriptions of both RT-11 and TSX-Plus EMTs; these are useful in conjunction with the SET EMT TRACE command. Appendix E contains listings of common subroutines called by the example programs throughout this manual. Appendix F contains explanations of monitor error messages; fatal system error messages are covered in the TSX-Plus System Manager's Guide. Appendix G describes the commands and techniques used to work with logical subset disks. Appendix H describes job execution priorities. Appendix I describes the commands and use of the TSX-Plus symbolic program debugger.

1. BASIC OPERATION

1.1 Logging on

Each user communicates with the TSX-Plus system through a time-sharing line (often referred to in this manual simply as a "line"). Lines are normally started by typing a carriage return at the terminal. Although, the system manager may designate lines to automatically initiate time-sharing whenever the system is started. In either case, when each line is first started, a greeting message will be displayed at the terminal.

The system manager may assign a start-up command file to be executed whenever a time-sharing line is started. A start-up command file contains initialization commands for a line. It can end by either starting execution of some program, or by waiting for a system command. (The keyboard monitor prompt is a period.) It is possible for a start-up command file to lock a program to a line so that on termination of the program the line is automatically logged off and an optional log-off command file is executed. Typing control-C will not abort start-up or log-off command files.

The system manager may also require "log on" authorization. In this case, the system manager assigns each user a user name (and project, programmer number) and password. After the greeting message, the message "Logon please:" is printed. The user responds by typing the user name (or project, programmer number) followed by carriage return. TSX-Plus then requests the password. The password is not echoed to the terminal as it is typed. After the user name (or project, programmer number) and password are validated, TSX-Plus types the message "Welcome to the system". The system manager may also designate a "start-up" command file for each account to control system access and set certain operating parameters.

The following example illustrates a typical log-on sequence. The information typed by the user is underlined.

(carriage return pressed)

()

(greeting message)

()

23-Jan-84 13:39:50

Line #2

Logon please:JOHNSON

Password:MYPASS (Password is not displayed.)

Welcome to the system

. (TSX-Plus is now waiting for a system command.)

A user may adopt a new password while logging on. To do this, enter a slash and the new password immediately after typing the old password. The new password must then be used for future logons. Passwords may be from 1 to 7 characters in length and must be composed only of letters and digits. The following example shows the password being changed from "OLDP" to "NEWP". Note that neither the old nor the new password would actually be echoed.

Basic Operation

(carriage return pressed)

()

(greeting message)

()

20-Jun-83 14:17:43

Line #5

Logon please:107,24

(Either user name or PPN may be used.)

Password:OLDP/NEWP

(Passwords are not displayed.)

Welcome to the system

. (TSX-Plus is now waiting for a system command.)

If the system manager has not required "log on" account authorization, then the system will either execute a start-up command file or simply display the monitor prompt (".") and wait for a system command to be entered.

1.2 Logging off

The OFF keyboard command is used to terminate ("log off") a time-sharing session.

The system manager may specify a "log off" command file to be executed whenever a job logs off. Control-C will not abort a "log off" command file.

1.3 Control characters

Certain control characters have special meaning to the system. These "control characters" control certain terminal operations. They are entered by holding down the "CTRL" key while pressing the selected character.

CTRL-C - Interrupts the current program and returns control to the keyboard monitor. If the running program is not waiting for input, two successive CTRL-C's are required to interrupt its execution. CTRL-C will not interrupt start-up or log-off command files.

CTRL-D - May interrupt the execution of the program and transfer control to the symbolic debugger. This depends on whether the debugging facility has been included during system generation and whether the SET CTRLD DEBUG command has been issued. See the description of the SET CTRLD DEBUG command and Appendix I.

CTRL-O - Suppresses program output to the terminal until one of the following conditions occurs:

- 1) A second CTRL-O is typed
- 2) The program returns to the monitor
- 3) The running program issues a .RCTRL O EMT

CTRL-Q - Resumes printing on the terminal from the point at which printing was previously suspended by CTRL-S.

CTRL-R - Causes the current characters in the terminal input buffer to be displayed. This can be used to check the actual contents of an input line when rubout editing has been done on the line.

CTRL-S - Temporarily suspends output to the terminal until CTRL-Q is typed.

CTRL-U - Deletes the current input line.

CTRL-W - Used to switch to a TSX-Plus virtual line. However, the virtual line switch character may be redefined during system generation. See Chapter 4 for more information on virtual lines.

CTRL-Z - Indicates end-of-file when input is being read from device "TT".

The normal function of these keys may be altered during program execution. Chapter 6 describes program controlled terminal options that influence these functions. When enabled, the single line editor defines additional special purpose keys; see the next section for further information about the single line editor.

Basic Operation

1.4 Single Line Editor

It is often desirable to correct typing mistakes made in the entry of command input lines or to correct data entry fields while executing some program. This can be done to a limited extent by use of the DELETE and CTRL-U keys, however much more editing capability is available if the "Single Line Editor" facility is used. To use the single line editor, the system manager must enable it during the TSX-Plus system generation process. In addition, it must be turned on for each time-sharing line before use. To use the single line editor, issue the command:

SET SL ON

either in a start-up command file or as a keyboard command. The single line editor may be used only with VT200, VT100 or VT52 type terminals.

Several other SET options are available for use with the single line editor. These are described in Chapter 2 in the section on keyboard commands.

The single line editor accepts special key commands to direct its operation. Under TSX-Plus, there are two modes of operation of the single line editor: normal mode (RT-11 compatible); and KED mode. In normal mode, the TSX-Plus single line editor is compatible with the RT-11 single line editor except for the differences noted in the table at the end of this section. The following tables summarize the functions performed by the editing control keys.

Single Line Editor Key FunctionsNormal (RT-11 compatible) Mode

Primary Key Functions (Not prefixed by PF1)	
Key	Function
Up arrow	Retrieve previous input lines
Down arrow	Retrieve line from save buffer
Left arrow	Move cursor left one character
Right arrow	Move cursor right one character
BACK SPACE	Exchange char under cursor with char to right
DELETE	Delete character to left of cursor
LINE FEED	Delete word to left of cursor
RETURN	Pass current line to running program
PF1	Used before other keys to perform second function
PF2	Not implemented
PF3	VT52: Delete from cursor to right end of line
PF4	VT100: Delete from cursor to right end of line
Ctrl-U	Delete from cursor to left end of line
Ctrl-R	Redisplay current line

Secondary Functions (Keys Prefixed by PF1)	
Key	Function
Up arrow	Retrieve previous input lines (PF1 is ignored)
Down arrow	Save current line in "save buffer" for recall later
Left arrow	Move cursor to left end of line
Right arrow	Move cursor to right end of line
BACK SPACE	Exchange char under cursor with char to left
DELETE	Retrieve last deleted character
LINE FEED	Retrieve last deleted word
RETURN	Truncate from cursor to end of line and execute
PF1	Ignored
PF2	Not implemented
PF3	VT52: Retrieve last deleted line
PF4	VT100: Retrieve last deleted line
Ctrl-U	Retrieve last deleted line

Basic Operation

In addition to the normal (RT-11 compatible) mode of operation for the single line editor, TSX-Plus also permits use of some of the keypad functions in a fashion similar to KED (or K52). These keypad operations are available in addition to the functions described above for the single line editor. In order to use the extended (KED) mode with the single line editor, issue the following command:

SET SL KED

When the KED mode is enabled, the following tables describe the additional functions available to the single line editor.

Single Line Editor Key Functions

KED Mode

Primary Key Functions (Not prefixed by PF1)	
Key	Function
0	Move cursor to left end of line
1	Move cursor one word in current direction
2	Move to end of line in current direction
3	VT100: Move cursor one char in current direction
4	Set direction forward (left to right)
5	Set direction backward (right to left)
6	VT52: Delete character under cursor
7	Not implemented
8	Not implemented
9	VT52: Delete word under cursor
-	VT100: Delete word under cursor
,	VT100: Delete character under cursor
ENTER	Pass current line to running program

Secondary Functions (Keys Prefixed by PF1)	
Key	Function
0	Delete from cursor to right end of line
1	Change case of char under cursor
2	Delete from cursor to right end of line
3	Not implemented
4	Move cursor to right end of line
5	Move cursor to left end of line
6	VT52: Retrieve last deleted character
7	Not implemented
8	Not implemented
9	VT52: Retrieve last deleted word
-	VT100: Retrieve last deleted word
,	VT100: Retrieve last deleted character
ENTER	Truncate from cursor to end of line and execute

To return the single line editor to the normal mode, issue the command:

SET SL NOKED

To disable the single line editor altogether, issue the command:

SET SL OFF

In addition to editing command lines, the single line editor may be used to edit data entry fields during program execution. However, there are some restrictions on such use. If any of the following conditions exist, then the single line editor may NOT be used to edit program data fields:

- 1) The SET SL OFF command has been issued
- 2) Bit 4 (EDIT\$, mask 20) is set in the Job Status Word (JSW)
- 3) Bit 12 (TTSPC\$, mask 10000) is set in the JSW
- 4) The program is using high-efficiency terminal mode
- 5) The program is using escape sequence activation
- 6) Input is being accepted via the .TTYIN EMT (except as below)

Basic Operation

In order to use the single line editor to edit data entry fields within programs which accept input via the .TTYIN EMT, the following command must be issued prior to execution of the program:

SET SL TTYIN

Since COBOL-Plus programs accept terminal input via the .TTYIN EMT, the following steps must be used to edit data entry fields while executing COBOL-Plus programs:

- 1) SET SL ON
- 2) SET SL TTYIN
- 3) From within the executing program, CALL "ESCAPE-OFF"

The TSX-Plus single line editor is generally compatible with that provided with RT-11, with the following exceptions:

- 1) The "Help" key (PF2) is not implemented.
- 2) The SET SL LEARN mode is not implemented.
- 3) The SET SL ASK option is ignored. (The current TSX-Plus terminal type information is used.)
- 4) The SET SL SYSGEN option is ignored.
- 5) The maximum width of a command line or input field that is accepted is 80 characters. The SET SL WIDTH=n option is ignored.
- 6) The line feed key deletes the word to the left of the cursor. Technically, characters to the left of the cursor are deleted until one of the following delimiters is reached: space, tab, comma, or equal sign.
- 7) The up arrow key can be used to retrieve either of the last two lines. Pressing the up arrow key cycles between recalling the last input line and the line prior to that.
- 8) Using the single line editor with .TTYIN input does not force a new line.
- 9) The single line editor is implemented as a TSX-Plus overlay region and does not require the SL pseudo-device handler.

2. KEYBOARD COMMANDS

2.1 Keyboard command interpretation

When a system command line is typed in, TSX-Plus attempts to interpret it by sequentially applying the following rules:

1. If the User Command Interface is in effect, then each time TSX-Plus is ready to accept a keyboard command it passes control to the current UCI program. The UCI program may be selected with the SET KMON UCI[=filnam] command. See the description of SET KMON command and the section in this chapter on the User Command Interface for more information.
2. If the command line begins with an "at-sign" ("@"), TSX-Plus attempts to execute a command file. If no device is specified with the command file name, device "DK:" is assumed. The default extension for command files is "COM". If the SET KMON IND command has been issued, command files will execute under the control of the IND program. Command files may be specified for execution as normal command files, regardless of whether KMON is set IND or NOIND, by starting the command with a dollar-sign (e.g. "\$@filnam"). Conversely, command files can be forced to execute under the IND utility, regardless of whether KMON is set IND or NOIND, by typing: "IND filnam".

See Chapter 3 for further information on the execution of command files by TSX-Plus.

3. If user-defined commands have been allowed during TSX-Plus generation, and if UCL is set FIRST either in TSGEN or subsequently by the keyboard SET UCL FIRST command, then commands are checked at this point to see if they are user-defined commands. However, if the first character on a command line is the underline character (" ") then the command on that line is not checked to see if it is a user-defined command. See the next section for more information on user-defined commands.
4. TSX-Plus next attempts to identify a command as a standard system command such as COPY, RUN, EXECUTE, etc. Commands may be abbreviated to the minimum number of characters that uniquely specify the name. Thus "COP" would be an acceptable abbreviation for COPY, but "CO" would not because it could mean COPY or COMPILE. "COPX" also would not be identified as a system command.
5. If user-defined commands have been allowed during TSX-Plus generation, and if UCL is set MIDDLE either in TSGEN or subsequently by the keyboard SET UCL MIDDLE command, then commands are checked at this point to see if they are user-defined commands. However, if the first character on a command line is the underline character (" ") then the command on that line is not checked to see if it is a user-defined command.

Keyboard Commands

6. If the command has not yet been identified, then TSX-Plus tries to find a command file on device "DK:" that has the same name as the command keyword. If no such command file is found on "DK:", TSX-Plus looks for the command file on device "SY:". In either case, if the SET KMON IND keyboard command has been issued, it will be executed under control of the IND program. When a command file is started in this fashion (rather than explicitly specifying an at-sign before its name), the command file listing is suppressed as if an implied "SET TT QUIET" command was executed during command startup. This implied listing suppression is temporary in effect and applies only to the command file started in this fashion. See Chapter 3 for more information on command files.
7. If a command file cannot be found, TSX-Plus looks for an executable program (SAV file) on device "SY:" that has the same name as the command keyword. If such a program is found, it is executed as if there were an "R" command in front of the program name. Note that while both RT-11 and TSX-Plus allow a line of text input to be passed to a program with the RUN command by specifying it after the program name, TSX-Plus also allows this to be done with the "R" command and with an implied "R" command line when the program name is specified as the command keyword. See example 6 below.
8. If user-defined commands have been allowed during TSX-Plus generation, and if UCL is set LAST either in TSGEN or subsequently by the keyboard SET UCL LAST command, then commands are checked to see if they are user-defined commands at this point. However, if the first character on a command line is the underline character ("_") then the command on that line is not checked to see if it is a user-defined command.
9. If a command has not been identified after all the above steps have been tried, then it is reported as an unrecognizable command. The error message is:

?KMON-F-Unrecognizable command

The following list summarizes the sequence of events which occur in the processing of keyboard commands. If a command can be recognized at any of these steps, then the appropriate command is executed and the remaining steps are skipped.

```

Call user command interface if one is active.
If command begins with "@", execute command file on DK:.
If UCL FIRST, check for user-defined command.
Check for system command.
If UCL MIDDLE, check for user-defined command.
Check for command file on DK:.
Check for command file on SY:.
Check for SAV file on SY:.
If UCL LAST, check for user-defined command.
Report unrecognizable command.
    
```

If user-defined command support has not been included in TSGEN, or if the SET UCL NONE command has been issued, or if a command line begins with the underscore character (" _"), then the command interpreter never attempts to interpret a command as a user-defined command. Attempts to issue user-defined commands under these conditions will also be reported as unrecognizable commands.

Examples:

1. Execute the command file "DK:PURGE.COM".
.@PURGE
2. Run a program named "DUMP" on device "SY:".
.R DUMP
3. Run a program named "PAYROL" on device "DX1:".
.RUN DX1:PAYROL
4. List all files on "RK1:".
.DIR RK1:
5. Start a command file "SY:LOADIT.COM" and pass it the parameter string "PROG2".
.LOADIT PROG2
6. Execute the program "SY:PIP.SAV" and pass it the input line "A.TMP=B.TMP".
.PIP A.TMP=B.TMP

Keyboard Commands

2.2 User-defined Commands

It is possible to define your own keyboard commands in terms of system commands and other keyboard commands. New keyboard commands may be defined according to the following syntax:

```
name ::= string
```

where "name" is a 1 to 11 character command keyword which may consist of letters, digits, and the underscore symbol ("_"), and "string" is a string of up to 80 characters which defines the body of the command.

For example, the following command would define the keyword "NOW" as being equivalent to the TIME command:

```
NOW ::= TIME
```

Multiple commands may be included in the body of a command definition by separating them with the backslash character ("\"). For example:

```
NOW ::= DATE\TIME
```

An up-arrow character ("^") may be included in the command body to cause all text on the command line following the command keyword to be inserted in the command body at the position of the up-arrow. For example, if a command is defined as follows:

```
NAMES ::= DIR/ORDER:NAME ^
```

Then, if this command is invoked by typing:

```
NAMES *.MAC
```

The resulting command will be equivalent to:

```
DIR/ORDER:NAME *.MAC
```

More than one up-arrow character may occur in the command body. The parameter string specified when the command is invoked is substituted for each occurrence of the up-arrow character.

A user defined command may invoke other user-defined commands within its definition provided that the definition does not call on itself and provided that the other commands are defined at the time the command is invoked. For example, the following command definitions would be legal:

```
NOW ::= SHOW DATE\SHOW TIME  
STATUS ::= NOW\SYSTAT
```


But, the following pair of commands would cause an infinite loop:

```
ONE == TWO
TWO == ONE
```

It is possible to allow abbreviation of user-defined commands. To do this, place an asterisk character ("*") within the keyword at the point of the minimum length abbreviation. For example, the definition:

```
ST*ATUS == NOW\SYSTAT
```

Would allow the STATUS command to be abbreviated to "STAT" or "ST" but not to "S".

You can specify the order in which the TSX-Plus command interpreter checks for user-defined commands by use of the SET UCL command. This command has four forms:

```
SET UCL FIRST
SET UCL MIDDLE    (This is the recommended setting.)
SET UCL LAST
SET UCL NONE
```

If the SET UCL FIRST command is used, user-defined commands will be processed before system commands. This allows user-defined commands to replace system commands but makes the processing of system commands slower. This is the required setting if it is necessary to replace some system commands.

If the SET UCL MIDDLE command is used, user-defined commands are processed after system commands but before checking for command files and SAV files with names that match the command keyword. Using this setting, it is not possible to replace a system command with a user command, but both system commands and user-defined commands are processed relatively quickly. This is the recommended setting unless it is desirable to replace system commands.

If the SET UCL LAST command is used, a command will not be checked to see if it is a user-defined command until after it has been checked to see if it is a system command, the name of a command file on DK, the name of a command file on SY, or the name of a SAV file on SY. Using this setting, it is not possible to replace a system command with a user command and user commands cannot have the same name as command files or SAV files. System commands are processed quickly (the same speed as SET UCL MIDDLE), but the processing of user-defined commands is slow. This is the appropriate setting only if user-defined commands are desired, but command files already exist whose names would conflict with user-defined commands. Existing command files which are short and merely execute system commands should be replaced by user-defined commands.

If the SET UCL NONE command is used, user-defined commands are never interpreted. In this mode, attempts to invoke user-defined commands will result in the error:

Keyboard Commands

?KMON-F-Unrecognizable command

The following list illustrates where the FIRST/MIDDLE/LAST setting causes the command interpreter to check for and process user-defined commands:

```
FIRST  -->      See if command is a system command
MIDDLE -->      Look for command file on DK: with command name
                Look for command file on SY: with command name
                Look for SAV file on SY: with command name
LAST   -->
```

See the beginning of this chapter for further information on the command interpretation process.

If an underscore character ("_") is typed in front of a keyboard command, this is a signal to the system that the command is not to be interpreted as a user-defined command. For example, if the NOW command has been defined as shown above, then the following command will be recognized as a user-defined command and will cause the date and time to be displayed:

```
NOW
```

However, the following command would not be recognized as a user-defined command and would be rejected as an undefined command since there is no NOW command defined by the system (unless there is a command file or SAV file on SY: with the name NOW):

```
_NOW
```

There are two reasons for using the underscore prefix. If UCL has been set FIRST, the underscore prefix can be used to speed up the processing of large numbers of system commands that could be present in frequently executed command files. For example, the NOW command could be defined as follows to cause it to execute faster:

```
NOW :== _DATE\_TIME
```

The second and more important reason for using the underscore prefix is to allow user-defined commands to be defined which replace system commands but which use the system commands in their definitions. (Note that it is necessary to SET UCL FIRST to allow system command replacement.) For example, the following definition replaces the system DIRECTORY command with a user-defined command that has the same name but which always orders the files alphabetically:

```
DIR*ECTORY :== _DIR/ORDER:NAME ^
```


Keyboard Commands

If the body of this command were not preceded with the underscore character, then it would be a circular definition and cause a futile loop.

A user-defined command can cause a command file to be executed. However, the command file invocation must be the last (or only) command defined within the body of the command. For example, the following definition causes all BAK files to be deleted, and a command file named QUIT to be executed when the OFF command is used (the QUIT command file could end with an "_OFF" command to do the actual logoff):

```
OFF ::= _DEL *.BAK/NOQ\@QUIT
```

A user-defined command may be replaced at any time by simply entering a new definition for the command. A command may be deleted by entering its name and "==" without a command body. For example, the following command deletes the definition of the NOW command:

```
NOW ==
```

The SHOW COMMANDS keyboard command may be used to display a list of all current user-defined commands.

Commands defined by one time-sharing user are "local" to that user and do not affect other users. However, when a virtual line is started the virtual line "inherits" the user-defined commands that are in effect for the primary line at the time that the virtual line is started. User-defined commands created while on a virtual line are not available from other virtual lines or from the primary line. All user-defined commands for a job are reset (forgotten) when the job logs off.

The system manager must enable user-defined commands by setting the U\$CL flag in TSGEN and the program TSXUCL.SAV must be on SY: in order to process user-defined commands. The maximum number of commands which can be defined by any job is set by the UCLMNC command in TSGEN. The default order for interpretation of user-defined commands during command processing is determined by the TSGEN parameter UCLORD. This may be overridden by the keyboard command SET UCL {FIRST|MIDDLE|LAST|NONE} for individual jobs.

Keyboard Commands

2.3 User Command Interface

The User Command Interface (UCI) allows a user-provided program to take over the job of command acquisition from the TSX-Plus keyboard monitor. When UCI is enabled, the user-written program will be called by the TSX-Plus keyboard monitor each time it is ready to accept a new command. It is then the responsibility of the user-written program to prompt for a command and accept it from the terminal. The program may then perform the appropriate actions, chain to other programs or pass commands on to the keyboard monitor. This provides a mechanism for such applications as a command menu.

UCI is enabled with the command:

```
SET KMON UCI[=filnam]
```

which may be entered either from the keyboard or in a start-up command file. If the optional "=filnam" is omitted, then the keyboard monitor passes command control to the program SY:UKMON.SAV. This is appropriate when a common command interface is desired for multiple lines. If different command interfaces are desired for different lines, then the file name of the appropriate user-written command interface should be specified.

After UCI is enabled, the TSX-Plus keyboard monitor will run the user-written UCI program each time it needs a new command. The program must prompt the user for a new command, accept the command, process it as desired and may optionally pass the command to the TSX-Plus keyboard monitor by doing a "special chain exit". A special chain exit is performed by issuing the .EXIT request with bit 5 (mask 40) set in the job status word and with R0 cleared. Any commands to be executed by the keyboard monitor are passed through the chain data area. See the RT-11 Programmer's Reference Manual for more information on "special chain exits". Commands passed to the TSX-Plus keyboard monitor in this fashion behave as though the keyboard monitor obtained them from a command file. A command file name may also be passed to the keyboard monitor by passing a command of the form "@name". If a command file name is passed to the keyboard monitor, then it must be the last or only command passed in the chain data area. When a command file name is passed in this manner, then all of the commands included in the command file are executed by the keyboard monitor before returning to the user-written UCI program for another command.

Keyboard command control may be returned to the TSX-Plus keyboard monitor by the command:

```
SET KMON SYSTEM
```

The following program provides a simple example of the techniques for writing a User Command Interface program. This program accepts a command from the keyboard and passes it through to the TSX-Plus keyboard monitor if it is a legal command.

Example:

```

        .TITLE  MYKMON
        .ENABL  LC

;
; Simple example of User Command Interface
; Refuses to pass SET KMON SYSTEM, but otherwise does nothing but
; pass commands thru to KMON.
;
        .MCALL  .PRINT,.EXIT,.GTLIN,.SCCA

JSW      = 44                ;Job status word address
SPXIT$   = 40                ;Special exit flag to pass command to KMON
MONPTR   = 54                ;Pointer to base of RMON
SYSGEN   = 372               ;Offset into RMON of SYSGEN options word

BEL      = 7                 ;ASCII bell
BS       = 10                ;ASCII backspace
LF       = 12                ;ASCII line feed
FF       = 14                ;ASCII form feed
CR       = 15                ;ASCII carriage return
ESC      = 33                ;ASCII escape

        .DSABL  GBL          ;Disable undefined globals
START:   .SCCA  #AREA,#TTSTAT ;Inhibit control-C abort
        MOV     @#MONPTR,R0   ;Get pointer to base of RMON
        TST     SYSGEN(R0)    ;Are we running under TSX?
        BPL     QUIT          ;Normal exit if not
        MOV     #TTYTYPE,R0   ;Point to EMT arg block to
        EMT     375           ;Get TSX-Plus terminal type
        ASL     R0            ;Convert to word offset
        CMP     R0,#4         ;Legal types are unknown, VT52 and VT100
        BLOS    1$
        CLR     R0            ;If not VT52 or VT100, make unknown
1$:      MOV     R0,R1         ;Save terminal type
        .PRINT  CLRSCR(R1)    ;Clear the screen
        .PRINT  #MENU        ;Display simple menu
        MOV     #SETRUB,R0    ;Point to EMT arg block to
        EMT     375           ;Set rubout filler character
2$:      .PRINT  CENTER(R1)    ;Move to screen center and clear the line
        .GTLIN  #BUFFER,#PROMPT ;Accept input line
        CALL    MATCH         ;See if it's legal
        BCS     2$            ;Repeat if illegal command
        MOV     #1000,SP      ;Ensure stack pointer safe
        CALL    MOVCMD        ;Move command from buffer to chain data area
        BIS     #SPXIT$,@#JSW ;Set special chain exit bit in JSW
        CLR     R0            ;Required for special chain exit
QUIT:    .EXIT                ;And pass command to KMON

; Simple matching. Easy to defeat by inserting extra spaces!!!

```


Keyboard Commands

```

MATCH:  MOV    #BUFFER,R2      ;Point to beginning of input buffer
        MOV    #ILLCMD,R3     ;Point to beginning of illegal command
1$:     TSTB   (R2)            ;At end of input string?
        BEQ    2$             ;Yes, matched so far, probably illegal
        CMPB   (R2)+,(R3)+     ;No, test through end of illegal string
        BNE    9$             ;No match, not illegal command
        CMP    R3,#ILLEND     ;Past end of illegal command?
        BLO    1$             ;No, keep checking
2$:     SEC                    ;Strings match, signal illegal command
        BR     10$
9$:     CLC                    ;Strings don't match, signal legal command
10$:    RETURN

```

; Move command from input buffer to chain data area.

```

MOVCMD: MOV    #BUFFER,R2      ;Point to beginning of input string
        MOV    #512,R3        ;Point to chain data area
1$:     MOVB   (R2)+,R0        ;Get next char
        BNE    2$             ;Continue if not nul
        CLRB   (R3)+          ;If end of input command
        BR     9$             ; then done
2$:     CMPB   R0,#'\          ;Command separator?
        BNE    3$             ;No, move it
        CLRB   R0             ;Yes, replace with nul
3$:     MOVB   R0,(R3)+        ;Move command into chain data area
        CMP    R2,#BUFEND     ;Don't want to overflow
        BLO    1$             ;Keep moving if characters left
        CLRB   -1(R3)         ;Mark end of command (ensure it is ASCIZ)
9$:     SUB    #512,R3         ;How many bytes did we move?
        MOV    R3,@#510       ;Mark the number for .CHAIN
        RETURN

```

```

AREA:    .BLKW   10            ;GP EMT argument area
TTSTAT:  .WORD   0             ;Terminal status word for .SCCA
TTYTYPE: .BYTE   0,137        ;EMT arg block to get terminal type
SETRUB:  .BYTE   0,152        ;EMT arg block to control terminal funtions
        .WORD   ^A            ;Function code - set rubout filler
        .WORD   ^             ;Rubout filler = underline
CLRSCR:  .WORD   CLRUNK,CLR52,CLR100 ;Terminal specific screen clears
CENTER:  .WORD   CNTUNK,CNT52,CNT100 ;Terminal specific move and clear
        .NLIST  BEX
CLRUNK:  .BYTE   FF,FF,FF,CR,200 ;Emulate clear screen with 3*(8LFs)
CLR52:   .BYTE   ESC,^H,ESC,^J,200 ;VT52 clear screen sequence
CLR100:  .ASCII  <ESC>/[H/<ESC>/[J/<200> ;VT100 clear screen sequence
CNTUNK:  .ASCII  <CR><LF><LF><LF>/ /<200>
CNT52:   .ASCII  <ESC>/Y% / ;Line 6, column 1
        .ASCII  <ESC>/K/ ;Erase to end of line
        .ASCII  / /<200> ;Move to column 6
CNT100:  .ASCII  <ESC>/[6;6f/ ;Line 6, column 6

```


Keyboard Commands

```

MENU: .ASCII <ESC>/[2K/<200> ;Erase entire line
      .ASCII <LF><LF><LF>/ ***** Simple Menu *****/<200>
PROMPT: .ASCII <BEL>/Command: _____/
      .NLIST
      .REPT 32.
      .BYTE BS ;Backspace to beginning of field
      .ENDR
      .LIST
      .BYTE <200> ;End of string
ILLCMD: .ASCII /SET KMON SYSTEM/ ;Don't permit UCI disable
ILLEND:
BUFFER: .BLKB 81. ;Command line input buffer
BUFEND:
      .END START

```


Keyboard Commands

2.4 Keyboard Commands

The keyboard commands accepted by TSX-Plus are listed below. Because many commands are identical or very similar to those of RT-11, full descriptions are only provided for TSX-Plus specific commands and for differences between TSX-Plus and RT-11 commands. Users should consult the RT-11 System User's Guide for more information on keyboard commands.

The ACCESS Command

The ACCESS command is used to restrict user access to a particular set of files or devices. It is only valid in start-up command files. Logical device names may be used in lieu of physical device names. Refer to the TSX-Plus System Manager's Guide for further information about this command.

The ALLOCATE Command

The ALLOCATE command is used to temporarily request exclusive access to a device. The form of the ALLOCATE command is:

```
ALLOCATE ddn[,ddn...] [xxx]
```

where "ddn" is the name of an allocatable device and "xxx" is an optional logical name to be assigned to the first device in the allocation list. Note that more than one device may be allocated with this command, but only the first may also be assigned a logical name in the same command. The device names "ddn" may also be substituted by previously assigned logical names. If an error occurs on the allocation of one device, an error message is printed and the system will continue to attempt to allocate any other specified devices. The most commonly used form of this command specifies a single device to be allocated and does not include a logical assignment. For example:

```
.ALLOCATE MTO:
```

If no device names are specified, the command is equivalent to SHOW ALLOCATE.

While a device is allocated by one user, access to that device is restricted exclusively to the user who allocated it; no one else will be allowed to open files on that device, to mount the device for directory caching, or to mount logical subset disks on that device. Conversely, a device cannot be allocated while any other user has that device mounted or a file open on that device. The maximum number of devices which may be simultaneously allocated by all users combined is determined during system generation. All devices allocated by a job are deallocated when that job logs off. See the DEALLOCATE command for more information on device deallocation. The SHOW ALLOCATE command may be used to determine which devices are currently allocated by any user.

Keyboard Commands

All jobs associated with a primary time-sharing line (the primary line and any virtual lines) have access to a device allocated by any other job associated with the same primary line. When a job connected to a primary line allocates a device, it overrides any allocation of the same device by a virtual job associated with that primary line. If two virtual lines associated with the same primary line attempt to allocate the same device, then the first allocation takes precedence.

Devices which are partitioned into multiple units may be allocated on a unit-by-unit basis. For example, allocating DL1 has no effect on use of DL0 by others; similarly, allocating CL0 has no effect on CL1. Only real devices may be allocated (those which are serviced by device handlers and CL). You cannot allocate the system device (SY:), the pseudo-device TT:, or any logical subset disks (LDn:).

Device allocation is useful when performing operations which have no other convenient mechanism for preventing disruptive interaction by multiple users. For example, if CL0 is attached to a communication port, undesirable confusion can arise if two users attempt to use the VTCOM program through CL0 at the same time. This can be overcome by allocating CL0 before running the VTCOM program. Another use for device allocation might be to prevent excessive rewinding delays when copying multiple files from magnetic tape while another user also attempts to access the same tape unit. It might even be desirable to allocate a removable device while it contains sensitive information.

The following example lists a command file which shows how the ALLOCATE command can be used to prevent conflicts in use of a communication line:

```
SET CLO LINE=6      !Take over time-sharing line
SET CLO NOLFOUT     !Inhibit line feed transmission
ASS CLO XL          !Assign for VTCOM
ALLOCATE XL         !Prevent multi-user collisions
R VTCOM             !Part of RT-11 5.01 distribution
DEALLOCATE XL       !Let others use CLO
DEASS XL            !Clean up assignment
SET CLO LINE=0      !Re-enable time-sharing line
```

The ASSIGN Command

The ASSIGN command is used to associate a logical I/O device name with a physical device. To simply assign a logical device name to a physical device, the form of the ASSIGN command is the same as that under RT-11:

```
ASSIGN ddn lnm
```

or

```
ASSIGN ddn=lnm
```


Keyboard Commands

where "ddn" is the physical device and unit specification (or another logical name previously assigned to a physical device) and "lnm" is a logical name (up to three characters long) by which the physical device may be referenced.

For example, to assign the logical device name "BIN" to physical device "DX1" the command would be:

```
ASSIGN DX1 BIN
```

The following command would assign logical device "BIN" to a file named "PROG1" on the system device:

```
ASSIGN SY:PROG1=BIN
```

It is also possible under TSX-Plus to assign a new logical name to a previously assigned logical name. The effect is to assign the new logical name to the same physical device to which the previous assignment was directed. For example, the following sequence of ASSIGNS result in both logical devices "AA" and "BB" being assigned to "DL1".

```
ASSIGN DL1 AA  
ASSIGN AA BB
```

The ASSIGN command is frequently used to assign FORTRAN I/O logical unit numbers to selected devices. To assign FORTRAN I/O logical unit number 1 to the terminal, the command would be:

```
ASSIGN TT 1
```

The TSX-Plus ASSIGN command provides a useful extension. In addition to being able to specify a physical device name, the user may specify a file name, extension, and size. If a file name and optional size are specified in addition to the physical device name, the file name and size follow the device name. For example, the following command assigns FORTRAN I/O logical unit number 1 to a file named "PAYROL" on device "DX0" with a size of 43 blocks:

```
ASSIGN DX0:PAYROL[43]=1
```

A maximum of fifteen assignments may be in effect at any given time for each user.

The BACKUP Command

The TSX-Plus BACKUP command has the same form and options as the RT-11 BACKUP command.

The BOOT Command

The BOOT command attempts to abort TSX-Plus and reboot RT-11. Due to the variations in hardware and bootstrap ROMs, this command is unsupported. The BOOT command may either reboot RT-11 on your system or simply halt depending on your particular hardware configuration. Unlike the RT-11 BOOT command, no device or file name may be specified with the TSX-Plus boot command; BOOT always attempts to reboot from the system (SY) device. Operator command privilege is required to use this command. The BOOT command is equivalent to the \$STOP command.

The BYE Command

The BYE command is used to log off a timesharing line. It is equivalent to the OFF command.

The COBOL Command

The COBOL command is used to compile a COBOL source program using the COBOL-Plus compiler. COBOL-Plus, a product of S&H Computer Systems, Inc., is sold separately. The default extension for COBOL source programs is "CBL"; the default extension for COBOL object files is "CBJ". The COMPILE, LINK and EXECUTE commands may also be used to compile and execute COBOL programs. TSX-Plus will implicitly invoke the COBOL-Plus compiler and CBLINK link program if the source program has the extension "CBL" or the object program has the extension "CBJ". Switches that can be used with the COBOL command are listed below.

Switch	Meaning
-----	-----
/ALLOCATE:size	Specify size of list or object file.
/ANSI	Produce warning messages for non-ANSI feature use.
/CARD	Source program is in card sequence format.
/CREF	(Equivalent to /CROSS).
/CROSS	Produce a cross-reference of the source program.
/DCARDS	Compile lines with "D" in the indicator field.
/INFORMATION	Print additional information at compilation end.
/LINENUMBER	Enable source line information in object listing.
/LIST[:name]	Produce a source program listing.
/NARROW	Format the cross-reference for 80 column display.
/OBJECT[:name]	Specify name of object file.
/ONDEBUG	Compile the program for use with the debugger.
/PRODUCTION	Omit line number tracing and subscript checking.
/RM	Compile RM/COBOL(*) programs.
/SEQUENCE	(Equivalent to /CARD).
/SUMMARY	Print only error messages on listing device.
/WARN	Suppress warning messages.

Keyboard Commands

See the COBOL-Plus reference manual for further information about the COBOL command.

* RM/COBOL is a trademark of Ryan-McFarland Corporation.

The COMPILE Command

The COMPILE command invokes the appropriate language processor to compile the specified source file. The TSX-Plus COMPILE command is the same as the RT-11 COMPILE command except that it also recognizes programs with the extension "CBL" as COBOL source programs and calls the COBOL-Plus compiler. When compiling a COBOL program, the switches that are legal with the COBOL command may also be used with the COMPILE command. It is also possible to explicitly specify that the COBOL-Plus compiler is to be called by using the "/COBOL" switch with the COMPILE command. The default compiler for files with the extension "DBL" is DBL; this may be changed with the SET LANGUAGE command.

The COPY Command

The TSX-Plus COPY command has the same form and options as the RT-11 COPY command.

The CREATE Command

The TSX-Plus CREATE command has the same form and options as the RT-11 CREATE command.

The DATE Command

The TSX-Plus DATE command has the same form and options as the RT-11 DATE command. Operator privilege is required to set the date.

The DEALLOCATE Command

The DEALLOCATE is used to release a temporary exclusive access restriction initiated by an ALLOCATE command for a device. This allows other users to again access the specified device. This command has three forms:

```
DEALLOCATE ddn[,ddn...]
DEALLOCATE                      }
DEALLOCATE/ALL                  } (equivalent commands)
```


Keyboard Commands

where "ddn" is the name of the device to be released. Logical device names may be used in lieu of the physical device names. Individual devices may be deallocated with the first form of the command. Individual devices allocated from the primary line or any other virtual line associated with the same primary line may be deallocated from the primary or any other virtual line associated with the same primary line. The second and third, non-specific, forms of the command are equivalent. All devices allocated by a job may be deallocated with a non-specific form of the command. However, this only affects devices allocated from the same primary or virtual line issuing the DEALLOCATE command. Devices allocated from other lines, even those associated with the same primary line, are not deallocated by non-specific forms of the DEALLOCATE command.

When a virtual line logs off, any devices allocated from that same virtual line are automatically deallocated, while those allocated from the primary line or a different virtual line are not deallocated. When a primary line logs off, any devices allocated from it or any of its associated virtual lines are automatically deallocated.

Logical names assigned to a device by using the ALLOCATE command are not deassigned by the DEALLOCATE command. See also the ALLOCATE and the SHOW ALLOCATE commands.

The DEASSIGN Command

The TSX-Plus DEASSIGN command is equivalent to the RT-11 DEASSIGN command. It is used to dissociate a logical unit assignment.

The DELETE Command

The TSX-Plus DELETE command has the same form and options as the RT-11 DELETE command.

The DETACH Command

The DETACH command is used to initiate execution of a command file as a "detached" job, to abort a detached job or to check the status of a detached job. The system manager may restrict the use of this command.

The form of the command used to start a detached job is:

DETACH file

where "file" is the name of a command file which is to be started as a detached job. The default file extension is ".COM". If a free detached-job line is available, the system starts the command file and prints a message indicating the detached job line number used. Detached-job lines must be declared when

Keyboard Commands

TSX-Plus is generated. The DETACH command itself and detached command files do not inherit any logical device assignments. That is, the file specification for the command file to be started as a detached job with the DETACH command must specify a physical device name, not a logical device name. The command file being detached may not be in a logical subset disk. If no physical device name is specified in the DETACH command, the command file is assumed to be on the system device (SY:). The command file being detached will have no logical assignments (DK: --> SY:) and must either make its own assignments or use physical device names.

In the following example a command file named "SY:CRUNCH.COM" is started as a detached job.

```
.DETACH CRUNCH  
Job started on line #5
```

If the specified command file is not found on the system device, the start message will still appear, but the job is not actually started. With no input, the detached job then aborts. Any error message would have been sent to the detached line, but is ignored since terminal output is not sent to detached jobs. The result is that the detached job is not started and no warning appears.

Terminal output for detached jobs is normally discarded since they are not attached to any terminal. However, it is possible to collect the terminal output from a detached job by sending it to a log file. This can be done by using the SET LOG FILE=filnam command in the detached job command file. See the SET LOG command for more information on terminal output logging.

The form of the DETACH command used to abort a detached job is:

```
DETACH/KILL line-number
```

where "line-number" is the number of the line assigned to the detached job.

The form of the DETACH command used to check the status of a detached job line is:

```
DETACH/CHECK line-number
```

In response to this command, TSX-Plus will indicate whether a job is still executing on the line.

See Chapter 4 for more information about detached jobs.

The DIBOL Command

The DIBOL command is used to compile a DIBOL or DBL source program. The default compiler for programs with the extension "DBL" is DBL. This may be changed with the SET LANGUAGE command. The TSX-Plus DIBOL command has the same form and options as the RT-11 DIBOL command, except that the options /BUFFERING, /LOG, /PAGE and /TABLES are not supported. Because of differences between compiler switch options, DIBOL switches are not supported for use with DBL.

The DIFFERENCES Command

The DIFFERENCES command is used to compare two files. The TSX-Plus DIFFERENCES command has the same form and options as the RT-11 DIFFERENCES command.

The DIRECTORY Command

The TSX-Plus DIRECTORY command has the same form and options as the RT-11 DIRECTORY command.

The DISMOUNT Command

The DISMOUNT command has three functions: 1) it tells TSX-Plus to stop directory caching on a particular device; 2) it tells the system to stop doing data caching on a device; 3) it dissociates a logical subset disk from its assigned file. The form of the DISMOUNT command is:

DISMOUNT ddn

where "ddn" is the real or logically assigned name of the device.

If the device is a physical device or a logical name for a physical device, then the DISMOUNT command removes the current job's entry from the mount table for that device. If no other jobs have mounted the device, then directory and data caching for that device are stopped. Files on a physical device may still be accessed after it is DISMOUNTed, but access may be slower since the directory is no longer cached. Note however, that if a job accesses a device which it has not mounted, and another user INITIALIZes or SQUEEZEs that device, then reads will probably return garbage and writes will probably corrupt other files. ALWAYS MOUNT ANY DIRECTORY STRUCTURED DEVICE WHICH YOU INTEND TO USE!

The INITIALIZE and SQUEEZE operations cannot be performed on a device which is mounted by other users. If it is necessary to INITIALIZE or SQUEEZE a device which is mounted by other users, then they must first DISMOUNT that device. Remember that it is still possible for a job which has not mounted a device to access that device. After a device is INITIALIZED or SQUEEZEd, then the directory and data caches are cleared for that device and if it is still mounted, then caching is resumed. Caching is not resumed if all jobs have

Keyboard Commands

DISMOUNTed a device until some job reMOUNTs that device. The following information message is printed if a device is dismounted and the device is still mounted by other users:

?KMON-I-Device is still mounted by other users

The SHOW MOUNTS command may be used to determine which jobs have devices mounted.

In the case of logical subset disk assignments ("ddn" = LDO-LD7 or logical names assigned to them), the effect of the DISMOUNT command is to stop directory caching on the logical subset disk as described above and to remove the device from the logical subset disk tables. In this case, files on the logical subset disk are no longer accessible until the logical subset disk is re-mounted. This form of the command only affects the logical subset disks belonging to the user who issues the command. If another job has mounted the same logical subset disk, then the rules for physical devices with regard to INITIALIZE and SQUEEZE also apply to the logical subset disk.

The DISPLAY Command

The DISPLAY command is used within a command file or user-defined command to cause a line of text to be displayed on the terminal when the command is executed. This is useful in command files that are not being listed to keep track of progress through the command file. The form of the DISPLAY command is:

DISPLAY comments

where "comments" can be any text string to be displayed on the terminal. See the section on user-defined commands in this chapter and see Chapter 3 for more information on command files.

The DUMP Command

The TSX-Plus DUMP command has the same form and options as the RT-11 DUMP command.

The EDIT Command

The TSX-Plus EDIT command has the same form and options as the RT-11 EDIT command. The editor which is invoked by this command is selected during system generation or with the SET EDIT command.

The EXECUTE Command

The TSX-Plus EXECUTE command has the same form and options as the RT-11 EXECUTE command. It also recognizes programs with the extension "CBL" as COBOL source programs and automatically invokes the COBOL-Plus compiler and linker. Switches appropriate when using the COBOL-Plus compiler and linker are also valid with EXECUTE. See also the COBOL, COMPILE, DIBOL, FORTRAN, LINK and MACRO commands.

The FORM Command

The FORM command is used to specify the default form name for subsequent files sent to the spooler by the user. The form of the FORM command is:

FORM name

where "name" is the one to six character default form name to be used for all files sent to the spooler until another FORM command is issued. The initial default form name is "STD". A form may also be requested within a file sent to the spooler. See Chapter 5 for more information on spooled devices and forms.

In the following example a FORTRAN listing will be generated for printing on a form called "2-PART".

```
.FORM 2-PART
.COMPILE/LIST TEST.FOR
.FORM STD
```

The FORTRAN Command

The TSX-Plus FORTRAN command has the same form and options as the RT-11 FORTRAN command.

The HELP Command

The TSX-Plus HELP command has the same form and options as the RT-11 HELP command.

The INITIALIZE Command

The TSX-Plus INITIALIZE command has the same form and options as the RT-11 INITIALIZE command. However, the system device (booted device when TSX-Plus is started) may not be initialized when running TSX-Plus. If it is necessary to initialize the system device, it must be done under RT-11. The resulting error message is:

Keyboard Commands

?KMON-F-This operation not legal with SY (system) device

A device cannot be initialized if any other user has allocated the device. Attempts to INITIALIZE a device allocated by another user will result in an error message like:

?KMON-F-Device is allocated to job 14

The SHOW ALLOCATE command may also be used to determine the job numbers which have allocated any devices. Note that a device allocated from another virtual job started from the same primary line is not protected in this manner.

A device cannot be initialized if any other user has MOUNTed the device. Attempts to INITIALIZE a device which is mounted by another user will result in the error message:

?KMON-F-Device is mounted by another user

If the device is only mounted by the job which initializes it, then the directory and data caches are cleared after the operation and caching is resumed. The SHOW MOUNTS command may be used to determine which other users have mounted a device. However, remember that is still possible for a job which has not mounted a device to access that device. If such a job has a file open before you initialize, then severe problems can arise. Great circumspection is necessary before initializing a device in a multi-user environment. Do not initialize rashly. As a defensive measure, always MOUNT any disk device which you plan to use.

If terminal output logging is being done (see the SET LOG command) and the log file is open on the device being initialized, then the log file is closed before the device is initialized and the following warning message appears:

?KMON-W-Closing log file

The KILL Command

The KILL command is used to abort a time-sharing job on another line. This has the effect of aborting the execution of the job and forcing the logoff of the line. Operator command privilege is required to use the KILL command. The form of this command is:

KILL line-number

where "line-number" is the number of the job to be killed.

The KJOB Command

The KJOB command is used to log off a timesharing line. It is equivalent to the OFF command.

The LIBRARY Command

The TSX-Plus LIBRARY command has the same form and options as the RT-11 LIBRARY command. It also can be used to build COBOL-Plus object program libraries; see the COBOL-Plus Reference Manual for further information.

The LINK Command

The TSX-Plus LINK command has the same form and options as the RT-11 LINK command. It also recognizes object files with the extension "CBJ" as COBOL-Plus object files and then invokes the COBOL-Plus link program (CBLINK). The COBOL-Plus linker may also be explicitly specified with the "/COBOL" switch. See the COBOL-Plus Reference Manual for further information. Switches which are unique to COBOL-Plus are:

Switch	Meaning
/NOPAGE	Do not swap data segments.
/NOSHARED	Do not use shared COBOL-Plus run-time library.
/SHARED	Always use shared COBOL-Plus run-time library.
/SIZE	Report the size of the largest data segment.
/VM	Use VM for run-time and program segmentation.
/XM	Load entire program and run-time into extended memory.

The LOGOFF Command

The LOGOFF command is used to log off a time-sharing line. It is equivalent to the OFF command.

The MACRO Command

The TSX-Plus MACRO command has the same form and options as the RT-11 MACRO command.

The MAKE Command

The MAKE command is used to create a new file with the TECO editor. It is equivalent to the RT-11 MAKE command.

Keyboard Commands

The MEMORY Command

The MEMORY command is used to control the amount of memory available to a job. When a job initially "logs on" it receives a default memory allocation set by the system manager. The MEMORY command can be used to change the allocation for the job. Use of the MEMORY command to change a job's memory allocation is not valid if the system has been generated to disallow job swapping. The form of the MEMORY command is:

MEMORY nn

Where "nn" is the number of k-bytes (Kb, 1024. bytes) of memory to be allocated for the job. The maximum memory size that a job may use is set by the system manager, but never exceeds 64Kb. When a running program performs a .SETTOP EMT the top of memory address corresponds to the size last specified by a MEMORY command. Note that .SETTOP EMT's do not actually affect the amount of memory allocated to a job -- only the MEMORY command and a TSX-Plus EMT described in Chapter 7 do that.

If the MEMORY command is entered without specifying a size, the current and maximum memory allocation for the job is displayed. See also the description of the SHOW MEMORY command.

Programs are only allowed to use more than 56Kb of memory if they are "virtual", meaning they do not directly access the RMON area, although they may still access it indirectly by use of the .GVAL and .PVAL EMT's. Programs may indicate that they are virtual by any of the following techniques:

1. Set bit 10 (mask 2000) in the job status word (location 44) of the SAV file. See Appendix A for information about how the SETSIZ program can be used to do this.
2. Use the /V LINK switch (/XM switch for the LINK keyboard command) which stores the RAD50 value for "VIR" in location 0 of the SAV file.
3. Store a memory allocation size greater than 56Kb in location 56 of the SAV file. See Appendix A for information about how the SETSIZ program can do this.

If none of these conditions is met the program is restricted to 56Kb even if a larger value is specified with the MEMORY command. See Chapter 8 and Appendix A for more information on memory usage and virtual images.

A program may dynamically control the amount of memory allocated for the job by use of the TSX-Plus EMT with function code 141 (described in Chapter 7). See also the description of the SETSIZ program in Appendix A for information about how the amount of memory to be allocated for a particular program can be stored in the SAV file for the program.

The MONITOR Command

The MONITOR command is used to cause TSX-Plus to begin a performance analysis. See Chapter 13 for complete information about the TSX-Plus performance analysis feature. The form of the MONITOR command is:

MONITOR base-address,top-address[,cell-size]/switches

where "base-address" is the lowest address in the program region being monitored, "top-address" is the highest address in the region, and "cell-size" is the number of bytes to group per histogram cell. The only valid switch is "/I" which causes I/O wait time to be included in the analysis.

The MOUNT Command.

The MOUNT command is used to: 1) begin directory caching on a file-structured device; 2) enable data caching on a device; and 3) associate a logical subset disk with a disk file.

Directory caching is a technique that speeds up file "lookups" by keeping information about files in memory so that it is not necessary to access the directory on the device each time a file is opened. Data caching is a technique used to speed up disk reads by keeping memory resident copies of recently used file blocks. Both directory and data caching are enabled during TSX-Plus system generation and activated when a device is MOUNTed. The form of the MOUNT command to activate directory and data caching is:

MOUNT ddn

where "ddn" is a physical device name such as "DL1:". The effect of this type of MOUNT command is to tell TSX-Plus that it should begin directory and data caching for the device being mounted. The system device is automatically MOUNTed for each user. If caching is not wanted, then no MOUNT should be performed, or the DISMOUNT command should be used to halt caching on a previously mounted disk. If the device being mounted is allocated by another user, then the system will report an error similar to:

?KMON-F-Device is allocated to job 12

The INITIALIZE and SQUEEZE operations cannot be performed on a device which is mounted by other users. If it is necessary to INITIALIZE or SQUEEZE a device which is mounted by other users, then they must first DISMOUNT that device. Remember that it is still possible for a job which has not mounted a device to access that device.

Once a MOUNT command is issued, caching is enabled for all users who access files on the device (including users who have not MOUNTed the device). The system maintains a table of all users who have mounted each device. See also the DISMOUNT and SHOW MOUNTS commands.

Keyboard Commands

Warning: If directory caching is enabled for a device, it is crucially important that the DISMOUNT command be used to dismount the device before a disk is replaced on the same drive. If a new disk pack is inserted in the drive without issuing the DISMOUNT command, TSX-Plus would try to access files on the new pack according to the locations stored in the directory cache for the old pack.

Directory caching causes a dramatic improvement in the speed of file "lookups" but does not speed up file "enters", "deletes" or "renames". This is because TSX-Plus always updates the directory on the device when it is altered. The maximum number of devices whose directories may be cached and the number of file entries that are kept in the directory cache are specified when TSX-Plus is generated.

The second form of the MOUNT command is used to associate a logical subset disk with a disk file. The form of the command is:

```
MOUNT[/[NO]WRITE] LDn filnam [logical-name]
```

where "LDn" is the logical subset disk, and n is in the range 0-7, "filnam" is the name of a disk file containing the subset directory and files, and "logical-name" is an optional logical name to be assigned to the logical subset disk. The [NO]WRITE option controls access to the logical subset disk. WRITE allows full access, whereas NOWRITE allows read-only access. WRITE access is allowed by default unless the NOWRITE option is used. The default extension for the disk file to be associated with a logical subset disk is "DSK". File protection is automatically set on the file specified to be a logical subset disk. When a logical subset disk is mounted, directory and data caching are also begun for it. If the device which holds the file to be mounted as a logical subset disk is allocated to another job, then the following error will be reported:

```
?KMON-F-File not found
```

Each user may mount up to 8 logical subset disks at any time. The association between a logical subset disk (LD0-LD7) and a file is "local" to each user. For example, one user may associate her LD2 with file DL1:MYDISK.DSK while a different user associates his LD2 with file DL1:YORDSK.DSK.

Logical subset disks may be "nested", allowing one or more subsets to be defined within other subsets. However, if this is done, the MOUNT commands must be executed sequentially from outer-most to inner-most logical subset disk and the unit numbers must be assigned sequentially from LD0 to LD7.

Example:

```
.MOUNT LD0 MANUAL MAN  
.CREATE LD0:SUB.DSK/ALLOCATE:100.  
.MOUNT/WRITE LD2 MAN:SUB  
.INIT/NOQ LD2:
```


Keyboard Commands

Other commands which refer to logical subset disks are: DISMOUNT, SET LDn {CLEAN|WRITE|NOWRITE} and SHOW SUBSETS. The ACCESS command may also be used with logical subset disks in start-up command files. The restrictions on INITIALIZE and SQUEEZE operations also apply when another user has mounted the same logical subset disk.

Logical subset disk support is integral to the file management functions of TSX-Plus, and does not require the "LD" pseudo-device handler. Consequently, it is independent of the version of RT-11 supporting TSX-Plus. See Appendix G for more information on the use of logical subset disks.

The MUNG Command

The MUNG command is used to start a file of TECO commands. The MUNG command is equivalent to the RT-11 MUNG command.

The OFF Command

The OFF command is used to log off a time-sharing line, and to release a virtual line (see the discussion of virtual lines in Chapter 4). The accumulated connect time and CPU time used during the session are printed during the logoff processing. The BYE, KJOB and LOGOFF commands are synonyms for the OFF command. TSX-Plus automatically logs off dial-up lines if the telephone connection is broken. A special log-off command file may also be designated to execute whenever a job logs off; see the TSX-Plus System Manager's Guide for more information.

Example:

.OFF

Connect time=01:43:00 CPU=00:12:03

The OPERATOR Command

The OPERATOR command is used to send a message to the operator's console. The OPERATOR command works like the SEND command, but it is not necessary to know the line number of the operator's terminal. The form of the OPERATOR command is:

OPERATOR message

For example, to send a disk mount message to the operator:

.OPERATOR PLEASE MOUNT PAYROLL MASTER DISK ON RK1

Keyboard Commands

The PAUSE Command

The PAUSE command is used within command files (see Chapter 3) to temporarily suspend processing of the file. The form of the PAUSE command is:

PAUSE comments

where "comments" may be any string of characters. When a PAUSE command is encountered within a command file, the PAUSE command is printed on the terminal followed by ">>". Execution of the command file is suspended until carriage return is pressed. This gives the operator an opportunity to perform manual operations such as mounting disks or tapes.

The PRINT Command

The TSX-Plus PRINT command has the same form and options as the RT-11 PRINT command.

The PROTECT Command

The TSX-Plus PROTECT command has the same form and options as the RT-11 PROTECT command.

The R Command

The "R" command is used to start a program. The form of the command is:

R[/switch] filnam [input-data]

If no device name is specified with the program name, TSX-Plus attempts to find the specified program on "SY:". The amount of memory available to a program can be controlled with the MEMORY command, or size information can be included in its disk image. See the description of the SETSIZ program in Appendix A for more information about how the amount of memory allocated for a program may be controlled. See also Chapter 8 for more information on the operating environment for programs under TSX-Plus.

A line of input may be passed to a program by specifying it as part of the "R" command following the program name. If this is done the program will receive the text string as its first line of input and will receive control-C as its second line of input. (See example 2 below.) Note that only programs which accept input with the .GTLIN, .CSISPC, and .CSIGEN requests can accept input in this manner. Single character input (.TTYIN) does not normally accept data from a command line; however, the program can be invoked with a command file and accept all terminal input from the command file - see the section on command file control characters in Chapter 3. Note also that text passed on the command line will be reorganized if it contains multiple words separated by

spaces. The first word will be placed as the input to a CSI command string and the remainder of the line will be placed on the output side of the equal sign. (See example 7 below.)

The valid switches for the R (and RUN) command are:

/DEBUG	-- Run program under debugger control
/HIGH	-- Run program in high efficiency TTY mode
/IOPAGE	-- Run program with PAR 7 mapped to I/O page
/LOCK	-- Lock program to line
/NONINTERACTIVE	-- Run program in non-interactive mode
/SINGLECHAR	-- Run program with single-character activation

All switches can be abbreviated to a single character.

The /DEBUG switch causes the program being started to execute under control of the TSX-Plus program debugging facility. Note that the debugger is a system generation option which must be included in order to use the debugging facility (see your System Manager). Since the debugger does not share memory space with the program being debugged, there are no restrictions on the size of program which can be debugged. It is not necessary to link the debugger with the program being debugged, although a link map of the program is essential to make effective use of the debugger. When a program is started under the debugger, the program is loaded and control is passed to the debugger. At this point, the starting address of the program is loaded into R0 and the debugger is set in such a way that the ";G" command will begin execution of the program. Execution of a program under debugger control may be interrupted as though a break point had been hit by typing a control-D (see the SET CTRLD DEBUG command). Debugger commands are similar to those of RT-11 ODT. See Appendix I of this manual for more information on the TSX-Plus program debugger.

The /HIGH switch automatically enables the program to use high efficiency terminal I/O. This disables much of the character testing done during terminal operations and can increase terminal throughput. See the description of the "R" program controlled terminal option in Chapter 6 for more information on high efficiency terminal mode.

The /IOPAGE switch causes PAR 7 for a program (virtual addresses in the range 160000 to 177777) to be mapped to the physical I/O page. Use of this switch requires operator privilege. Under TSX-Plus a program usually has PAR 7 mapped to a simulated RMON; this allows old programs which directly access RMON fixed offsets to operate without being rewritten. However, when PAR 7 is mapped to a simulated RMON, access to hardware control registers in the I/O page is not allowed without special coding. When a program is started with the /IOPAGE switch, then direct I/O page access is possible without recoding; however, direct RMON access is not available. See Chapter 8 for more information on the TSX-Plus job environment. See Chapter 11 for more information on special system service calls which can be used to control job virtual address mapping, especially RMON vs. I/O page mapping.

Keyboard Commands

The /LOCK switch causes the program that is being started to be "locked" to the time-sharing line so that the line is automatically logged off when the program exits. If the "R/LOCK" command occurs within a command file the command file is terminated as the program is started and any additional information in the command file is ignored. The most frequent use of this feature is in start-up command files where a line is to be restricted to executing a particular program. If a locked program chains to another program, the program that was chained to then becomes the locked program.

The /NONINTERACTIVE switch prevents a program from receiving the priority boost normally given to jobs on the completion of terminal input. This should be used with programs which do heavy terminal I/O but which are not really interactive jobs, such as file transfer programs. A program run with this switch will execute at a lower priority and will not interfere with interactive jobs. The effect of this switch is cleared when a program exits to KMON, but the switch remains in effect if the program chains to another program.

The /SINGLECHAR switch causes a program to execute in "single character activation" mode. (See the discussion of activation characters in Chapter 6.) Normally when programs are run under TSX-Plus they do not receive terminal input until an "activation" character such as carriage-return has been entered. This is true even if the program sets bit 12 in the Job Status Word. Also, TSX-Plus does not normally allow a program to test for terminal input without stalling on the .TTYIN EMT. However, the /SINGLECHAR switch causes TSX-Plus to honor bits 6 and 12 of the Job Status Word, allowing the program to activate on each character and test for terminal input without stalling. A program can also cause TSX-Plus to honor JSW bits 6 and 12 by using the "U" and "S" terminal control commands (see Chapter 6). The example program "STEALS" in the section on requesting exclusive system control in Chapter 11 uses single character activation in this way. KED and K52 are automatically run in single character activation mode.

Examples:

1. Run the program named "DUMP" on device "SY:".

.R DUMP

2. Run the program named "PIP" on "SY:" and pass to it the input line "A.TMP=B.TMP. (If no system command, UCL command, or command file name conflicts with a program name, then a program on SY: may be run simply by name. See the beginning of Chapter 2 for more information on command interpretation.)

.PIP A.TMP=B.TMP

3. Run the program named "SAMPLE" on "RK2:".

.R RK2:SAMPLE

Keyboard Commands

4. Start the execution of BASIC and force logoff on exit.

.R/LOCK BASIC

5. Start a program named PLANE and allow it to use single character activation mode.

.R/SINGLE PLANE

6. Start a program named TRIAL in debug mode so that it will be run under TSODT.

.R/DEBUG TRIAL

TSX-Plus debugger

DBG:

7. Start the program SY:GETLIN and pass it some input text.

.R GETLIN INPUT OUTPUT

The program GETLIN will receive the following text:

OUTPUT=INPUT

The RENAME Command

The TSX-Plus RENAME command has the same form and options as the RT-11 RENAME command.

The RESET Command

The RESET command is used to reset the system usage statistics that are displayed with the SYSTAT command. Data caching statistics are also reset by the RESET command. This is useful when you want to monitor system performance during a particular part of the day. Operator command privilege is required to use the RESET command. The TSX-Plus RESET command is NOT equivalent to the RT-11 RESET command.

The RUN Command

The RUN command is equivalent to the "R" command except that the default device is "DK:" instead of "SY:". See the description of the "R" command for information about available switches.

Keyboard Commands

The SEND Command

The SEND command is used to send messages between time-sharing terminals. The form of the command is:

SEND[,line#] message

where "line#" is the number of the line to which the message is to be sent. If no line number is specified, the message is broadcast to all logged-on lines. Jobs may inhibit the reception of messages while executing programs by using the SET TT GAG command.

Examples:

1. Send a message to all logged-on users:

.SEND Bob, Call me when you get a chance.

2. Send a message to line number 2:

.SEND,2 Will you be on tonight?

When a SEND message is printed at a terminal, the message is preceded by the number of the line that originated the message and the user name currently associated with that line. For example, a message from line 1 might be printed as follows:

01 (SYSMGR) -- Bob, Call me when you get a chance.

Keep in mind that several characters are used to identify the sending line and user, with the result that a long message may be truncated.

The SET Command

The SET command is used to set various options controlling system operation. The general form of the SET command is:

SET device option

As with RT-11, the TSX-Plus SET command is used to specify options for devices such as line-printers and card-readers as well as setting certain system parameters such as terminal control characteristics. When used to set device options, the SET command has the same form as under RT-11 and may set the same options (they are specified in the device handler). The device name may be either the physical device name or a logical name assigned to the physical device. For example:

.ASSIGN CL1 XX
.SET XX LENGTH=66

The SET command may use logical names for physical devices and for CL, LD and VM devices, but not for pseudo-devices like TT and SL, nor for system parameters like QUAN1, CACHE or PRIORITY. The SET command causes the copy of the device handler on the disk to be altered so that the effect of the command becomes "permanent" (until another SET changes the parameter back). The SET command also attempts to make the change to the copy of the handler that is in memory with TSX-plus. If the handler is idle when the SET is issued, the change will be made; otherwise, a warning message:

?KMON-F-Handler active -- Can't update running copy

will be printed and the running copy of the handler is not altered. When a SET is done to a device handler, blocks 0 and 1 of the handler are read from the disk, the SET option is applied and then block 1 is written back to the disk and moved over the copy of block 1 that is in memory. If the vector of a device is changed with the SET dd VECTOR=nnn command, TSX-Plus must be restarted to function correctly. Operator command privilege is required to set an option in a device handler. See the RT-11 System User's Guide for device handler SET options.

SET CACHE

The SET CACHE command is used to alter the number of blocks which may be held in the generalized data cache. This command does not alter the amount of memory reserved for the data cache, but only controls the number of blocks within the limits allowed by the CACHE parameter selected during system generation. This command is used by the system manager to determine the effect of varying cache sizes on system performance. Operator privilege is necessary to use this command. The form of this command is:

SET CACHE blocks

where "blocks" may range from 0 to the number of blocks reserved by the CACHE parameter during system generation.

SET CCL

There are two types of system commands, low level commands such as RUN, SET, ASSIGN and high level commands such as EXECUTE, COPY, DELETE, and DIRECTORY. Low level commands are executed directly by TSX-Plus. High level commands are translated into the appropriate low level commands before execution. The set of high level commands is known as the Concise Command Language (CCL). The "SET CCL" command can be used to observe the low level commands that are produced by translating CCL commands. The form of this command is:

Keyboard Commands

SET CCL [NO]TEST

When TSX-Plus is in CCL TEST mode, it will display at the terminal the low level commands that are generated by a CCL command, but not execute them. The "SET CCL NOTEST" command turns this mode off and TSX-Plus resumes executing CCL commands. Test mode is very useful if you are having trouble getting some complex CCL command to work and want to examine the low level commands that are being generated.

Example:

```
.SET CCL TEST
.DIR/OUTPUT:DIR.DAT/OCTAL/BLOCKS DL1:
R DIR
DK:DIR.DAT=DL1:*/O/B
^C
.SET CCL NOTEST
```

SET CL

The SET CL command is used to control the functions of a CL (Communication Line) pseudo-device. The CL facility is a device handler which may be used to service most serial devices such as printers, plotters and dial-out lines. CL units may either be assigned to dedicated lines which can only be used as I/O devices or may be assigned to "take over" inactive time-sharing lines on a temporary basis. This might be useful when a particular line is to be used as a "dial in" time-sharing line at some times and as a "dial out" communication device at other times. The CL handler can support up to 8 units and can be used to replace the LS and XL device handlers. The CL handler can serve as a communication interface with virtual terminal or file transfer utilities, such as the RT-11 VTCOM program. CL lines, as well as time-sharing lines, may be connected through DH(V)11, DL(V)11 and DZ(V)11 type interfaces. See the TSX-Plus System Manager's Guide for specific programming information for the CL device handler.

SET options are issued for each individual CL unit. The number of available CL units is determined during TSX-Plus system generation. CL unit numbers range from zero to one less than the number of CL units defined during system generation. For example: if a system is generated with a total of 5 CL units, then the units would be designated CL0 through CL4. The unit CL: is equivalent to CL0:.

SET options for CL units are not permanent; they return to the default parameters and must be reset each time TSX-Plus is restarted. A detached job start-up command file is a convenient way to set CL options each time TSX-Plus is restarted. If an unattached CL unit is assigned to take over a time-sharing line, then that CL unit acquires the default options. The SHOW CL command may be used to determine current settings for any CL unit attached to a line.

Keyboard Commands

The SET CL command has two forms, one for on/off options that accept a NO prefix, and another for options that accept a value. Those which accept a NO prefix are used to toggle between two possible states. Those which accept a value are used to control variable features like page length. Examples of these two forms are:

```
SET CLO LINE=6
or
SET CL1 NOLFOUT
```

The SHOW CL and SHOW TERMINALS commands and the SYSMON utility may be used to obtain information about the current status of CL units.

Operator privilege is required to issue SET CL commands.

A typical use of CL is to redefine a "dial up" time-sharing line for temporary use as a "dial out" communication device using the RT-11 VTCOM utility. A typical command file to do this might be:

```
SET CLO LINE=6      !Take over time-sharing line
ASSIGN CLO XL       !Assign for VTCOM (use XC on the PRO)
ALLOCATE XL         !Prevent multi-user collisions
SET XL NOLFOUT      !Inhibit line feed transmission
R VTCOM             !Part of RT-11 5.01 distribution
!Communicate with remote system using VTCOM
DEALLOCATE XL       !Let others use CLO
DEASS XL            !Clean up assignment
SET CLO LINE=0      !Re-enable time-sharing line
```

A second common use for CL is to drive serial printers, as a replacement for the LS device handler. In some situations with fixed system resources it can be useful to alternate a single interface port between use as a time-sharing line and as a printer port. The printer port on a Professional 350 is one example. A typical command file to make this redefinition is:

```
SET CL2 LINE=2      !Redefine printer port as spooled CL unit
SET CL2 SPEED=1200  !1200 baud printer
SET CL2 LENGTH=66   !Set page length
SET CL2 NOFORM      !Printer does not have form feeds
SET CL2 FORM0       !Start new page for each file
ASS CL2 LP          !Assign so PRINT command works
```

These and all other available SET CL commands are explained in the following table.

Keyboard Commands

SET CLn

Option	Meaning
--------	---------

[NO]BININ	This option controls binary input mode. In binary input mode: 8 bit data characters are passed directly to the user's data buffer; received control characters (such as control-S, control-Q and control-Z) are treated as normal data characters; a read operation is only completed when the specified word count is satisfied. The default mode is NOBININ.
-----------	--

[NO]BINOUT	This option controls binary output mode. In binary output mode: 8 bit data characters are sent to the CL unit without modification; the CL handler does not send XON or XOFF characters to control the speed of the remote device, but it will respond to XON/XOFF characters received from the remote device unless binary input mode is also in effect. The default mode is NOBINOUT.
------------	---

[NO]CR	This option controls transmission of carriage return characters to the output device. If a CL unit is set NOCR, carriage return characters are discarded and not sent to the output device. Carriage return characters received from the CL device are not affected. The default setting is CR.
--------	---

[NO]CTRL	This option controls transmission of control characters to a CL output device. Control characters are those ASCII characters from NUL to US (octal values 0 to 37; decimal values 0 to 31). When a CL unit is set NOCTRL, control characters are not sent to the output device. This includes the ESC character. Carriage and paper movement characters are exceptions; BS, HT, LF, FF and CR are always transmitted, regardless of the setting of CTRL. (Note, however, that other parameters may affect transmission of these characters, such as the LFOUT and FORM settings.) Characters received from the CL device are not affected by the CTRL setting. The default setting is CTRL.
----------	---

[NO]DTR	This option causes the Data Terminal Ready (DTR) signal to be asserted on interface devices capable of modem control. NODTR causes the DTR signal to be dropped. The DTR signal is automatically asserted if a read or write operation is directed to a CL unit, even if that unit has been set NODTR. In this case, DTR remains asserted even after the operation has completed until it is specifically turned off. The default (initial) setting is NODTR.
---------	---

[NO]FORM	This option controls the transmission of form feed characters to CL output devices. If a device's hardware can utilize form feed characters, the CL unit should be set FORM. If a CL unit is set NOFORM, then form feeds sent to the unit are translated to enough line feed characters to advance to the top of the next form, based on the setting of the LENGTH parameter. The default setting of this parameter for dedicated CL units is determined during TSX-Plus
----------	--

Keyboard Commands

system generation. For extra CL units, the default setting is NOFORM.

[NO]FORM0 This option controls paging at the beginning of print files. More precisely, if a CL unit is set FORM0, then a form feed (or enough line feeds) will be sent each time a write is issued to the device with a block number of 0. The default setting is NOFORM0.

[NO]LC This option controls case conversion of output characters. If a CL unit is set NOLC, then lower case characters (a - z) will be translated to upper case (A - Z) before transmission. Characters received from the CL unit are never converted to upper case. The default setting is LC.

LENGTH=n This option defines the number of lines per page. This option is used together with the NOFORM option to replace form feed characters with enough line feeds to advance to the top of a page. If LENGTH=0 and NOFORM are selected, no form feed simulation occurs and form feeds are discarded. This option is also used with the SKIP option to define bottom margins. The default setting is LENGTH=66.

[NO]LFIN This option determines the fate of line feed characters received from a CL unit. If a unit is set NOLFIN, then line feed characters received from the CL device are discarded. The default setting is LFIN.

[NO]LFOUT This option controls the transmission of line feed characters to a CL device. If a unit is set NOLFOUT, then line feed characters sent to the CL device are discarded. When a CL unit is used with the RT-11 VTCOM utility, it should be set NOLFOUT. The default setting is LFOUT.

LINE=n This option is used to redirect a CL unit to use a time-sharing line or a dedicated CL line. The parameter "n" must refer to a physical time-sharing line number or to the line number assigned to a dedicated CL line. You may not assign a CL unit to a time-sharing line which is in use or to a dedicated CL line which currently has another CL unit assigned to it. A CL unit may be disassociated from any line by setting it to "LINE=0". Time-sharing lines may be temporarily redefined as output devices or "dial-out" ports by assigning a CL unit to them. They may then be restored as time-sharing lines by disassociating the CL unit from them. A CL unit may not be assigned to a line which is flagged as DEAD. The SHOW CL and SHOW TERMINALS commands may be used to determine the current assignments of CL units.

Keyboard Commands

- SKIP=n** This option is used together with the LENGTH setting to control bottom margins. When only "n" lines remain on a page, a form feed (or enough line feeds) is issued to advance to the top of the next form. This feature is normally used to skip over page perforations between forms. The default setting is SKIP=0, which does not skip lines at the bottom of a page.
- SPEED=n** This option controls the transmit/receive baud rate for a CL unit. Split speeds are not supported. This option can only be used for hardware interfaces which support programmable baud rates, such as DZ11, DZV11, DH11, DHV11, and DLV11-E interfaces and the Professional 350 printer and communication ports. The available speed values are: 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, and 19200. DEC DZ(V)11 interfaces do not support the 19200 baud rate. DEC DHV11 interfaces do not support the 7200 baud rate. DEC DH11 interfaces do not support the 3600 or 7200 baud rates.
- [NO]TAB** This option controls transmission of horizontal tab characters to CL units. If the device can support the tab character, the TAB setting should be used. If the unit is set NOTAB, then tab characters sent to the unit are replaced with enough spaces to advance to the next tab position. (Substituted tabs are at columns 1, 9, 17, 33, 41, ...) The default setting for dedicated CL lines is determined during TSX-Plus system generation. The default setting for extra CL units is NOTAB.
- TOP** This option is used to define the top of a page (form). This is necessary when repositioning forms on a printer which does not support hardware form feeds.
- WIDTH=n** This option specifies the maximum line width. Characters sent to a CL unit are discarded if they would exceed the maximum line width. The current column count is cleared after each carriage return. The default setting is WIDTH=0, which allows unlimited line widths.

SET CORTIM

The SET CORTIM command is used to adjust the value of the CORTIM system control parameter. This parameter controls the minimum memory residency time for jobs just swapped into memory. See the TSX-Plus System Managers Guide for further information about the CORTIM parameter. The form of this command is:

SET CORTIM value

where "value" is the time value specified in 0.1 second units. The current value of the CORTIM parameter may be determined with the SHOW CORTIM command. Operator command privilege is required to use this command.

SET CTRLD

The SET CTRLD command is used to enable or disable automatic entry to the TSX-Plus program debugger when control-D is typed. Any time a program is running under control of the debugger, then control-D may be used to interrupt the program execution and pass control to the debugger as if a breakpoint had been hit at the current location. Normally, this is only possible if the program was started under debugger control (/DEBUG switch to the R or RUN command). However, it is also possible to enable control-D to interrupt programs which were not started under the debugger and pass control to the debugger. To enable this feature, issue the command:

SET CTRLD DEBUG

To disable control-D interruption of programs which were not started under the debugger, issue the command:

SET CTRLD NODEBUG

See the description of the /DEBUG switch to the R command and the description of the debugger in Appendix I for more information.

SET EDIT

The SET EDIT command is used to select which edit program will be invoked when the system EDIT command is used. The form of this command is:

SET EDIT option

where "option" may be EDIT, TECO, KED or K52. The options KED and K52 are actually synonymous; the KED editor is used if the terminal type has been specified to be a VT100 or VT200 and the K52 editor is used if the terminal type has been specified to be a VT52. VT200 terminals must be set to 7-bit control characters to function with KED. The terminal must be SET TT LC in order to use KED or K52.

SET EMT

The SET EMT command is used to control tracing of EMT calls during the execution of a user program. The form of the command is:

SET EMT [NO]TRACE

When SET EMT TRACE is specified, a line of information about the EMT call is displayed at the terminal each time an EMT is executed. The .TTYIN, .TTYOUT and .PRINT EMTs, however, are not included in EMT traces. EMT tracing is disabled with the SET EMT NOTRACE command. Each line of information which is displayed during EMT tracing contains: the virtual address of the EMT call, the EMT code, function code, channel number (or sub-function code), and the

Keyboard Commands

first 5 words in the EMT argument block. Only the first two items are defined for all EMTs. The other items are defined only if used for the EMT currently being traced. As an example of EMT tracing, the LNTT program which displays the current line number and terminal type (see Chapter 7) was traced as follows:

```
.SET EMT TRACE  
.RUN LNTT
```

```
001004 374 004 000 000000 000000 000610 000000 000000
```

```
001012 375 110 000 057400 001252 001262 001270 001277
```

```
001026 374 005 000 057400 001252 001262 001270 001277
```

```
TSX-Plus line number: 2
```

```
Terminal type:
```

```
001060 375 137 000 001252 001262 001270 001277 001311
```

```
VT-100
```

```
001072 350 016 010 001270 001262 001270 001277 001311
```

```
.SET EMT NOTRACE
```

Note that the .PRINT calls are not traced, since .TTYIN, .TTYOUT and .PRINT EMTs are never traced. Appendix D contains a list of both RT-11 compatible and TSX-Plus specific EMTs.

SET ERROR

The SET ERROR command is used to specify the level of error which will abort command file execution. The form of this command is:

SET ERROR option

where "option" may be FATAL, SEVERE, ERROR, WARNING or NONE. Command files being executed under the IND program are not normally aborted if errors occur during execution; the SET IND ABORT command can be used to cause IND command files to abort according to the same rules as for normal command files. The TSX-Plus SET ERROR command functions in the same fashion as the RT-11 SET ERROR command.

SET HIPRCT

The SET HIPRCT command is used to set the value of the HIPRCT system control parameter. See the TSX-Plus System Manager's Guide for more information on the effect of this parameter. The form of this command is:

SET HIPRCT value

where "value" sets the non-interactive job I/O counter. Operator privilege is necessary to use this command. The HIPRCT parameter may be referenced by the SET SIGNAL and SHOW commands. Refer to the TSX-Plus System Manager's Guide for more information on job scheduling and performance optimization.

SET IND

The SET IND [NO]ABORT command is used to control the execution of command files under the control of the IND program when there is an error. Normally, command files under the control of IND are not aborted when an error occurs, regardless of the current SET ERROR level. However, if the SET IND ABORT command is issued, then command files under the control of the IND program will abort under the same conditions as would normal command files. The SET IND NOABORT command restores the default abort processing under IND control (no abort).

SET INTIOC

The SET INTIOC command is used to set the value of the INTIOC system control parameter. See the TSX-Plus System Manager's Guide for more information on the effect of this parameter. The form of this command is:

SET INTIOC value

where "value" is the interactive job I/O counter. Operator privilege is necessary to use this command. The INTIOC parameter may be referenced by the SET SIGNAL and SHOW commands. Refer to the TSX-Plus System Manager's Guide for more information on job scheduling and performance optimization.

SET IO

The SET IO [NO]ABORT command is used to select the method of handling I/O abort requests. If the SET IO ABORT command is issued, then an I/O abort request will call device handler abort entry points. If the SET IO NOABORT command is issued, then I/O abort requests will proceed through I/O rundown; that is, all pending I/O will complete before the job is aborted. The initial setting of this parameter is selected during TSX-Plus system generation. Operator privilege is required to use this command. The method selected affects all lines, not just the line from which the command is issued.

SET KMON

The SET KMON command is used to direct the processing of commands from the keyboard and from command files. The form of this command is:

SET KMON option

Keyboard Commands

where the valid options are: [NO]IND, UCI[=filnam] and SYSTEM. The chain of events in command processing by TSX-Plus is described at the beginning of this chapter.

The processing of indirect command files may either be controlled by TSX-Plus or by the IND utility provided with RT-11. To cause command files to be processed by IND, use the command: SET KMON IND. To return to the normal mode of TSX-Plus command file processing: SET KMON NOIND. Note that command files which result in errors are not normally aborted, regardless of the SET ERROR level, when executed under control of IND. The SET IND [NO]ABORT command can be used to control error abort of IND command files. Command files may be forced to the normal TSX-Plus mode of execution regardless of whether KMON is set IND or NOIND by calling them as:

```
$@filnam
```

Conversely, command files may be forced to execute under the IND utility regardless of whether KMON is set IND or NOIND by calling them as:

```
IND filnam
```

See Chapter 3 for more information on command file processing.

The other function of the SET KMON command is to control the user command interface. It is possible for user written programs to accept and pre-process keyboard commands before the TSKMON program. Command control is local to each user. That is, each job may select its own command processing method. The User Command Interface may be enabled by the command:

```
SET KMON UCI[=filnam]
```

When UCI is in effect, then each time TSKMON is ready to accept a command it passes control to the current UCI program. If the optional "=filnam" has been omitted, then TSKMON passes command acquisition control to the program SY:UKMON.SAV. If a file has been specified, then TSKMON passes control to that program. It is the responsibility of the user-written command interface program to prompt for and accept command input lines. Commands may be further passed on to TSKMON through the chain-data area by doing a special chain exit.

Command processing control is returned to TSKMON by the command:

```
SET KMON SYSTEM
```

See the example program in the section on the User Command Interface earlier in this chapter for more information.

SET LANGUAGE

The SET LANGUAGE command is used to select DBL or DIBOL as the default compiler for programs with the extension DBL. This also affects the COMPILE and EXECUTE commands. The form of the command is:

SET LANGUAGE option

where "option" is DBL or DIBOL. The default setting of this parameter is DBL.

SET LD

The SET LD command is used to control writing to a logical subset disk and to verify logical subset disk assignments. The form of the command is:

SET LDn option

where "LDn" is in the range of LD0 to LD7, and "option" may be CLEAN, WRITE or NOWRITE. To prevent writing to a logical subset disk, SET LDn NOWRITE. To allow writing to a logical subset disk, SET LDn WRITE. WRITE/NOWRITE control is also an option of the MOUNT command. SET LDn CLEAN is used to verify and correct logical subset disk assignments. An implicit SET LDn CLEAN is done by TSX-Plus whenever the DUP utility program is run (e.g. INIT and SQUEEZE operations). See the MOUNT command and Appendix G for further information on the use of logical subset disks.

SET LOG

It is possible to copy terminal output to a log file. When terminal logging is enabled, all output directed to the terminal is also written to the log file. The only exception to this is high-efficiency terminal output which is not logged. To initiate terminal logging issue the following command:

SET LOG FILE=name

where "name" is the file specification for the log file. The default extension is ".LOG". For example, the following command would copy terminal output to a file named "DK:RUNLST.LOG":

SET LOG FILE=RUNLST

The following command causes terminal output to be copied to the line printer:

SET LOG FILE=LP:

Logging of terminal output may be stopped and the log file closed by the command:

Keyboard Commands

SET LOG CLOSE

The log file is automatically closed when another log file is opened or the job logs off. The log file is also automatically closed if the device containing the log file is initialized or squeezed; the following warning message appears:

?KMON-W-Closing log file

It may be desirable to suspend and resume terminal output logging during some operations without closing and reopening another log file. To temporarily suspend terminal logging, issue the command:

SET LOG NOWRITE

To resume terminal logging, issue the command:

SET LOG WRITE

It may be desirable at some times to reset the terminal log file. To clear the contents of the log file without closing and deleting the file, issue the command:

SET LOG CLEAN

Terminal logging is especially useful with detached jobs. Terminal output to detached jobs is normally discarded since the job is not attached to any terminal. However, when terminal logging is enabled for a detached job, then the terminal output may be directed to a file or to a printer.

SET LOGOFF

The SET LOGOFF command is used to associate a "log-off" command file with a job. This command is only valid within start-up command files. The form of this command is:

SET LOGOFF FILE=name

where "name" is the file specification of a command file to be executed when the job logs off. Log-off command files, like start-up command files, cannot be aborted by control-C. See the TSX-Plus System Manager's Guide for further information about log-off command files.

SET MAXPRIORITY

The SET MAXPRIORITY command may be used to reduce the maximum priority allowed to a job. When placed in a start-up command file, this command restricts the maximum available priority for that job. This permits restriction of priority on lines which do not use the LOGON facility. The form of this command is:

SET MAXPRIORITY value

where "value" is in the range of 0 to 127. If the current maximum job priority is less than 127, then the current maximum priority is the upper limit for the valid range of "value". See the TSX-Plus System Manager's Guide for further information about restricting job priority.

SET NUMDC

The SET NUMDC command is used to control the number of buffers used for data caching. The form of this command is

SET NUMDC value

where "value" is the number of buffers to use. The initial value of the NUMDC parameter is specified when the system is generated. The SET NUMDC command may be used to restrict the number of data cache buffers actually used to a value less than the number of buffers specified when the system was generated, but may not exceed that value. The SET NUMDC command does not alter the memory space allocated for cache buffers, it merely controls the number of buffers actually used. Operator command privilege is required to use this command.

The primary use of the SET NUMDC command is to determine the optimum number of data cache buffers to include in a system. To do this, the system can be generated with a large number of data cache buffers and the SET NUMDC command can be used to determine the minimum number which are actually needed to provide effective performance. When using this procedure, all files that are being cached should be closed before the SET NUMDC command is issued. See Chapter 9 for a discussion of shared-file data caching.

SET PRIORITY

The SET PRIORITY command is used to set the execution priority for a time-sharing job. The form of the command is:

SET PRIORITY value

where "value" may range from 0 to 127. The default priority assigned to a job when not otherwise specified or restricted is 50. The maximum priority allowed to a job may be restricted through the logon mechanism or the SET MAXPRIORITY command. The SHOW PRIORITY command can be used to display the current job priority and the maximum authorized priority. The SYSTAT command also displays current job priorities.

Job priorities may also be assigned from within an executing program with an EMT. See Chapters 7 and 11 for more information on setting job priority from within a program.

Keyboard Commands

When a job is disassociated from the terminal by switching to a different virtual line the job is reduced in priority by an amount determined during system generation (the PRIVIR parameter).

The job priority value is used by the scheduler to determine which job has precedence and should be run next. The priority value is only used when there is more than one job in the same executable state queue. See Appendix H for more information about job execution priorities. See the TSX-Plus System Manager's Guide for more information on job scheduling.

SET PROMPT

The SET PROMPT command is used to change the keyboard monitor prompt character. The default character is a period ("."). The form of the command is:

SET PROMPT "string"

where "string" is a quoted string containing from 1 to 8 characters. The quoted string will subsequently be used as the keyboard monitor prompt string. For example, in order to set the monitor prompt to a dollar sign followed by a space (the standard VMS prompt string), use the following command:

SET PROMPT "\$ "

To restore the standard RT-11 prompt string, issue the command:

SET PROMPT "."

SET QUANxx

The SET QUANxx command is used to set the value of the system time-slice control parameters (QUANO, QUAN1, QUAN1A, QUAN1B, QUAN1C, QUAN2, and QUAN3). See the TSX-Plus System Manager's Guide for information about the effect of the these parameters. The form of this command is:

SET QUANxx value

where "value" is the time value specified in 0.1 second units. Operator command privilege is required to use this command. The value selected takes effect for all jobs on the system, not just the job issuing the command. These parameters may be referenced by the SET SIGNAL and SHOW commands. See the TSX-Plus System Manager's Guide for more information on job scheduling and performance optimization.

SET SIGNAL

The SET SIGNAL command is used as an aid to system performance tuning. The form of the SET SIGNAL command is:

SET SIGNAL [NO]parameter

where "parameter" may be one of the system scheduling parameters: HIPRCT, INTIOC, QUANO, QUAN1, QUAN1A, QUAN1B, QUAN1C, QUAN2, or QUAN3. Only one parameter may be selected by each SET SIGNAL command, although with multiple SET SIGNAL commands more than one parameter may be selected for signaling at once. Signaling may be disabled for any individual parameter by prefacing the parameter with "NO". Signaling may be halted for all parameters with the command SET SIGNAL OFF.

When signaling has been enabled for a system tuning parameter, the bell will be rung at the terminal of the job for which signaling has been selected each time the job changes state because it exceeds the value of the selected parameter. The SET SIGNAL command functions on a line-by-line basis and only affects the line from which the command is issued.

The signaling feature is intended as an aid to the system manager in determining appropriate values of the system tuning parameters for a given job. See the TSX-Plus System Manager's Guide for more information on use of the signaling feature.

SET SL

The SET SL command is used to control the ability to edit keyboard input lines. The TSX-Plus SL facility is generally compatible with that provided with the RT-11 "SL" editor. It allows editing of the current input line or field and recall of the previous input line or field. Logical names may not be used in lieu of "SL". See the section on the single line editor in Chapter 1 for information on the use of the single line editor. The form of this SET command is:

SET SL option

where the options are described below.

SET SL
Option

Meaning

ASK

This command is ignored. The terminal type is determined from the current TSX-Plus terminal type.

K52

This command enables "KED"-like extensions to the single line editor. Note that the numeric keypad is set to application mode.

Keyboard Commands

KED	This command enables "KED"-like extensions to the single line editor. Note that the numeric keypad is set to application mode.
KEX	This command enables "KED"-like extensions to the single line editor. Note that the numeric keypad is set to application mode.
[NO]LEARN	This command is ignored. The single line editor LEARN mode is not implemented.
[NO]LET	This command is ignored. The LET utility may not be used with the single line editor.
OFF	This command turns off the single line editor.
ON	This command turns on the single line editor. Note that the job is in single-character activation mode.
RT11	This command disables the "KED"-like extensions to the single line editor.
SYSGEN	This command is ignored. The single line editor is implemented as a TSX-Plus system overlay region and does not require the SL pseudo-device handler.
[NO]TTYIN	This command either enables (TTYIN) or disables (NOTTYIN) editing of keyboard input being accepted via the .TTYIN EMT from within programs.
VT52	This command is equivalent to SET TT VT52. Ensure that your terminal responds to VT52 type control sequences before issuing this command.
VT62	This command is equivalent to SET TT VT52. Ensure that your terminal responds to VT52 type control sequences before issuing this command.
VT100	This command is equivalent to SET TT VT100. Ensure that your terminal responds to VT100 type control sequences before issuing this command.
VT101	This command is equivalent to SET TT VT100. Ensure that your terminal responds to VT100 type control sequences before issuing this command.
VT102	This command is equivalent to SET TT VT100. Ensure that your terminal responds to VT100 type control sequences before issuing this command.

Keyboard Commands

VT200 This command is equivalent to SET TT VT200. Ensure that your terminal responds to VT200 control sequences before issuing this command. VT200 terminals should be set in VT200 mode, with 7-bit control characters.

WIDTH=n This command is ignored. The maximum input line width that can be used with the single line editor, either as a command line or in a program data field, is 80 characters.

SET TERMINAL

The SET TERMINAL command is used to set various terminal parameters. It is equivalent to the SET TT command.

SET TT

The SET TT command is used to set various terminal and job parameters, and is equivalent to the SET TERMINAL command. Logical names may not be used in lieu of "TT". The form of this SET command is:

SET TT [line-number] [NO]option

to turn an option off and on. For example:

.SET TT FORMO

Some options require a numeric parameter, in which case they are specified as:

SET TT [line-number] option=value

For example:

.SET TT 6 SPEED=9600

The optional "line-number" parameter refers to the TSX-Plus line number, which is determined by the order in which time-sharing lines were declared during system generation. The hardware interface to which each line is attached may be determined from the SHOW TERMINALS command. Only physical line numbers may be specified with the SET TT command, not detached or virtual lines.

If "line-number" is omitted from the command, the setting is applied to the terminal from which the command is issued. Operator privilege is required to issue a SET TT command for a line other than that from which the command is issued.

Keyboard Commands

If "line-number" is specified, then the setting becomes permanent and will remain in effect until changed or until the system is restarted. If "line-number" is omitted, then the setting is temporary and will be reset when the line logs off. Terminal SPEED setting is an exception to this rule. Line speed is additionally affected by the AUTOBAUD option. If a line is set AUTOBAUD, then a temporary change reverts to AUTOBAUD when the line logs off, whereas a permanent change clears AUTOBAUD and remains in effect across logoffs. If a line is not set AUTOBAUD, then any SPEED setting, either with or without "line-number" specified, is permanent and remains in effect across a logoff.

SET TT Option -----	Meaning -----
ADM3A	Tells TSX-Plus that the terminal being used is a Lear Siegler ADM3A and also has the effect of SET TT SCOPE, NOTAB, NOFORM.
[NO]AUTOBAUD	Allows automatic baud rate selection for terminals during log on. This is only functional for lines connected through interface cards which support programmable baud rates. The NOAUTOBAUD option disallows automatic baud rate selection. Use of the SET TT line-number SPEED command also cancels automatic baud rate detection. Operator privilege is required to issue this command for a line other than the one from which the command is issued.
[NO]DEAD	Disables further logons on the line. If a line is active when the command is issued, the line is not disabled until the job logs off. The NODEAD option reenables logons. A CL unit may not be assigned to a line flagged as DEAD. Operator privilege is required to issue this command.
DECWRITER	Equivalent to the LA36 option.
[NO]DEFER	TSX-Plus offers two modes of character echoing -- "deferred" and "immediate" (NODEFER). If the user only types input to programs while they are waiting for input, the two modes function identically. However, if the user types input before the program finishes processing the previous line of input, the two modes are different. In immediate (NODEFER) mode the input characters are echoed immediately and may be printed even before the program prints the response to the previous line. In deferred echo mode, the characters that are typed ahead are accepted and held for the program, but are not echoed to the terminal until the program is ready to accept them. Under standard RT-11, EDIT, BASIC, and DIBOL programs run in deferred mode. Most other programs use immediate echoing. Deferred echoing is the preferred mode under TSX-Plus. See Chapter 6 for information about how a program can control deferred echo mode.

Keyboard Commands

DIABLO	Has the effect of SET TT NOSCOPE, FORM, NOTAB, PAGE. The QUME type is equivalent. When doing plotting or printing with proportional spacing, SET TT TAB.
[NO]ECHO	Controls echoing of characters to the terminal. See Chapter 6 for information about how a program can control character echoing.
[NO]EIGHTBIT	Controls 8-bit I/O for a terminal line. When in 8-bit mode, all 8 bits of each character are passed to and from a terminal. The null character (000) may never be received from the terminal. In addition, in EIGHTBIT mode, if VTxxx escape sequences are defined as a terminal input activation condition, then the character 377 (octal) may not be received. In 8-bit mode, terminals should be set for 8-bit characters with no parity. VT2xx terminals should be set to VT200 mode with 7-bit control characters.
[NO]FORM	Controls conversion of form feed (FF) characters to line feeds. FORM should be set with terminals whose hardware can respond to form feed characters. NOFORM should be used for terminals whose hardware cannot handle form feed characters. When NOFORM is set, form feed characters are replaced by eight line feed characters.
[NO]FORMO	The FORMO option causes TSX-Plus to first issue a form-feed when a write is done to the terminal with a block number of zero. This is convenient when producing multiple program listings to cause each listing to begin at the top of a new page. The NOFORMO option disables special handling of block zero writes.
[NO]GAG	The GAG option inhibits messages sent from another line from being displayed at the terminal. This is only in effect when the job is executing a program. Messages are not inhibited while in the keyboard monitor. The NOGAG option allows messages to be displayed at any time. The GAG option is useful for preventing messages from interrupting jobs on hardcopy terminals, such as printers or plotters, where a message could spoil the format. This command only affects messages sent either with the SEND command or the EMT which sends a message to another job. This does not apply to message communication channels.
HAZELTIME	Tells TSX-Plus that the terminal being used is a Hazeltine brand terminal and also has the effect of SET TT SCOPE, NOTAB, NOFORM.
LA36	Tells TSX-Plus that the terminal being used is an LA36 and also has the effect of SET TT NOSCOPE, NOTAB, NOFORM.
LA120	Tells TSX-Plus that the terminal being used is an LA120 and also has the effect of SET TT PAGE, TAB, FORM, NOSCOPE.

Keyboard Commands

- [NO]LC Allows lower case characters to be passed to a program. If LC is set and bit 14 of the job status word is set to 1, input of lower case characters from the terminal will be passed to the running program. If the terminal is set NOLC or the job status word bit 14 is clear, lower case characters are translated to upper case. See Chapter 6 for information about how a program can control lower-case character conversion. Note that in order to use the keypad editors KED and K52 the terminal must be SET TT LC.
- [NO]PHONE Selects modem control. NOPHONE disables modem control, indicating that the line is hardwired to a terminal. See the TSX-Plus System Manager's Guide for more information on modem control under TSX-Plus. Operator privilege is required to issue this command.
- [NO]PRIVILEGE Enables operator privilege for a line. Operator privilege is required to issue certain commands which alter system behavior. The NOPRIVILEGE option disables operator privilege for a line. Operator privilege is required to issue this command. A privileged line can cancel its own privilege, but a non-privileged line cannot grant privilege. See the TSX-Plus System Manager's Guide for more information on operator privilege.
- [NO]QUIET Setting the terminal QUIET suppresses the listing of command files as they are executed. When the terminal is set NOQUIET, command file lines are listed as they are executed. See Chapter 3 for additional information on controlling command file listing.
- QUME Equivalent to the DIABLO type. Has the effect of SET TT NOSCOPE, FORM, NOTAB, PAGE. When doing plotting or printing with proportional spacing, SET TT TAB.
- [NO]SCOPE Tells TSX-Plus whether the terminal is a CRT device. Setting SCOPE causes the DELETE key to echo as a backspace-space-backspace sequence, erasing the previously typed character. See Chapter 6 for information about how a program can specify an alternate "rubout filler" character that will be used to overwrite characters being erased when DELETE is typed. When the terminal is set NOSCOPE, the DELETE key causes preceding characters to be echoed in reverse order as they are removed from the input buffer, and CTRL-U echoes carriage return, line feed. CTRL-R can be used to check the current contents of the input buffer when doing rubout editing. See Chapter 1 for information on other special control characters.

Keyboard Commands

- [NO]SINGLE Controls automatic single character activation for programs. If the SET TT SINGLE command has been issued, then programs may control single character activation on terminal input by toggling bit 12 in the job status word. The /SINGLE switch to the R[UN] command and the "S" program controlled terminal option also allow individual programs to control single character activation. The SET TT SINGLE command applies to all programs until the SET TT NOSINGLE command is issued. If the SET TT NOSINGLE command is in effect, then programs may still individually control single character activation with the R[UN]/SINGLE switch or the "S" program controlled terminal option. In all cases, the program must still set bit 12 in the job status word to achieve single character activation. The SET TT SINGLE command differs from the R[UN]/SINGLE switch in that it only affects setting of single character activation (JSW bit 12), not terminal no-wait input (JSW bit 6). The SET TT NOWAIT command may be used to allow no-wait terminal input.
- SPEED=n Specifies the transmit/receive speed for a terminal line. The available speed values are: 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, and 19200. This command is only functional for those lines connected through interface cards which support programmable baud rates (DLV11-E, DH(V)11, DZ(V)11 and the PRO-350 printer and communications ports). DEC DZ(V)11 interfaces do not support 19200 baud. DHV11 interfaces do not support 7200 baud. DH11 interfaces do not support 3600 or 7200 baud. SET TT line-number SPEED cancels automatic baud rate selection, which may be restored with the SET TT AUTOBAUD command.
- [NO]TAB Controls conversion of TAB characters to multiple spaces. TAB should be selected with terminals whose hardware can respond to TAB characters. When set NOTAB, TAB characters being sent to the terminal are replaced by an appropriate number of spaces.
- [NO]TAPE Causes TSX-Plus to ignore received line-feed characters. See Chapter 6 for information about how a program can turn tape mode on and off.
- VT50 Equivalent to the VT52 option.
- VT52 Tells TSX-Plus that the terminal being used is a VT52 (or VT100 in VT52 mode) and also has the effect of SET TT SCOPE, TAB, PAGE, NOFORM.
- VT100 Tells TSX-Plus that the terminal being used is a VT100 (which must be operating in VT100 mode -- not VT52 compatible mode) and also has the effect of SET TT SCOPE, TAB, PAGE, NOFORM.

Keyboard Commands

VT200 Tells TSX-Plus that the terminal being used is in the VT2xx series and also has the effect of SET TT SCOPE, TAB, NOFORM, EIGHTBIT. The terminal type may also be specified as VT220, VT240 or VT241, all of which are equivalent to VT200.

[NO]WAIT Normally, TSX-Plus blocks the execution of a program that does a .TTINR EMT if no activation character has been received even if the program sets bit 6 (inhibit terminal wait bit) in the Job Status Word which is supposed to mean that the program can do non-blocking .TTINR character tests. This is done to prevent programs from "burning up" CPU time by constantly looping back to test for terminal input. If the NOWAIT option is specified with the SET TT command, TSX-Plus will honor bit 6 in the Job Status Word and allow the program to do non-blocking .TTINRs if bit 6 is set. The example program "STEALS" in the section on requesting exclusive system control in Chapter 11 demonstrates the program controlled terminal option to allow NOWAIT input.

Most of these terminal options can be given initial settings for each line when the TSX-Plus system is generated. Each time a user logs onto a line, the initial option settings are used. The options may be altered by using the SET command, but when the user logs off, the options revert to their initial setting specified in TSGEN. When a user initiates a virtual line, the initial flag settings for the virtual line are copied from the current flag settings for the user. Subsequent flag changes for a virtual line do not affect flag settings for the user's other lines. The SYSMON utility may be used to determine the current settings for any time-sharing line.

SET UCL

You can specify the order in which the TSX-Plus command interpreter checks for user-defined commands by use of the SET UCL command. This command has four forms:

SET UCL FIRST
SET UCL MIDDLE
SET UCL LAST
SET UCL NONE

If the SET UCL FIRST command is used, user-defined commands will be processed before system commands. This allows user-defined commands to replace system commands but makes the processing of system commands slower. This is the required setting only if it is necessary to replace some system commands.

If the SET UCL MIDDLE command is used, user-defined commands are processed after system commands but before checking for command files and SAV files with names that match the command keyword. Using this setting, it is not possible to replace a system command with a user command, but both system commands and user-defined commands are processed relatively quickly. This is the recommended setting unless it is desirable to replace system commands.

If the SET UCL LAST command is used, a command will not be checked to see if it is a user-defined command until after it is checked to see if it is a system command, the name of a command file on DK, the name of a command file on SY, or the name of a SAV file on SY. Using this setting, it is not possible to replace a system command with a user command and user commands cannot have the same name as command files or SAV files. System commands are processed quickly (the same speed as SET UCL MIDDLE), but the processing of user-defined commands is slow. This is the appropriate setting only if user-defined commands are desired, but command files already exist whose names would conflict with user-defined commands. Existing command files which are short and merely execute system commands should be replaced by user-defined commands.

If the SET UCL NONE command is used, user defined commands are never interpreted. In this mode, attempts to invoke user-defined commands will result in the error:

?KMON-F-Unrecognizable command

The following list illustrates where the FIRST/MIDDLE/LAST setting causes the command interpreter to check for and process user-defined commands:

```

FIRST  -->      See if command is a system command
MIDDLE -->      Look for command file on DK: with command name
                Look for command file on SY: with command name
                Look for SAV file on SY: with command name
LAST   -->

```

See the beginning of this chapter for further information on the command interpretation process.

SET VM

The SET VM command is used to control the amount of memory available to the VM pseudo-device. The SET VM command is normally not necessary as VM will automatically calculate the correct base address to use, starting just above the last address used by TSX-Plus, and using all physical memory installed above that address. The only form of this command is:

SET VM BASE=nnnnnn

where "nnnnnn" represents the octal value of bits 6 through 22 of the base memory address which VM is allowed to use. For example, in a system with 1.5 megabyte of memory, to reserve the top 256 Kb of memory for use by VM the value of "nnnnnn" should be 50000. This corresponds to a base address for VM of 5000000. If this command is issued, then VM will use the higher of the address specified by "nnnnnn" or the top of TSX-Plus. That is, it is not permitted to set the base of VM to overlap memory regions reserved for use by TSX-Plus.

Keyboard Commands

SET WILDCARDS

The SET WILDCARDS command is used to control substitution of missing parts of file names. The form of the command is:

SET WILDCARDS option

where "option" is either IMPLICIT or EXPLICIT. If IMPLICIT is chosen, then file specifications which are incomplete will be treated as though the "*" character had been typed for the missing part of the specification. The only parts of file specifications affected by this substitution are the file name and the extension. The device cannot be specified as a wildcard. Most keyboard commands which accept file names also accept wildcards. The following table demonstrates the interpretation of various file specifications when wildcards have been set IMPLICIT or EXPLICIT.

Command	IMPLICIT	EXPLICIT
-----	-----	-----
cmd TEST.FOR	cmd TEST.FOR	cmd TEST.FOR
cmd TEST.*	cmd TEST.*	cmd TEST.*
cmd TEST	cmd TEST.*	cmd TEST
cmd TEST.	cmd TEST.	cmd TEST.
cmd *.FOR	cmd *.FOR	cmd *.FOR
cmd .FOR	cmd *.FOR	cmd .FOR
cmd *.*	cmd *.*	cmd *.*
cmd *	cmd *.*	cmd *
cmd .*	cmd *.*	cmd .*
cmd	cmd *.*	cmd

where "cmd" is one of the keyboard commands which accepts wildcards in file specifications, such as: COPY, DIRECTORY, PRINT, PROTECT, etc.

The system-wide default setting for wildcard interpretation may be selected in TSGEN; see the TSX-Plus System Manager's Guide.

The SHOW Command

The SHOW command is used to display information about the state of the system. Each form of the SHOW command is described below.

SHOW ALL

The SHOW ALL command is equivalent to specifying all of: SHOW DEVICES, SHOW ALLOCATE, SHOW ASSIGNS, SHOW JOBS, SHOW TERMINALS, SHOW CL, SHOW MEMORY, SHOW SPOOL, SHOW SUBSETS, SHOW MOUNTS, SHOW RUN-TIMES.

SHOW ALLOCATE

The SHOW ALLOCATE command displays a list of all devices allocated for exclusive use by any job on the system. For example:

.SHOW ALLOCATE

Device	Job	User name
-----	---	-----
DB5	7	MIMSY
MT0	15	BOROGROVES

SHOW ASSIGNS

The SHOW ASSIGNS command displays information about all logical device assignments that are currently in effect.

Example:.SHOW ASSIGNS

Assignments:

SY --> DL0:

CBL --> RK0:

TMP --> DL3:

DK --> LD0:

SHOW CACHE

The SHOW CACHE command reports the total number of blocks available in the generalized data cache buffer.

Example:.SHOW CACHE

Number of blocks in data cache = 1000

SHOW CL

The SHOW CL command is used to determine the current characteristics and assignments of all CL units included during TSX-Plus system generation. CL units are typically used either to control serial devices like printers and plotters or as communication lines to other computer systems. CL units may either be dedicated only for use as CL ports, or may be generated as extra units that can be used to take over inactive time-sharing lines or dedicated CL lines which do not have any currently assigned CL unit. CL units may also be spooled (see Chapter 5 for more information on device spooling). Several parameters may be set for each individual CL unit and the SHOW CL command is

Keyboard Commands

used to determine the current settings for each unit. See the SET CL command for descriptions of the various parameters which may be set for CL units. The SHOW TERMINALS command also indicates which lines are currently assigned as CL units. The "Line" column indicates which TSX-Plus line number each CL unit is assigned to. The "Job" column shows the line number of any job which is either currently using or has allocated each CL unit.

Example:

.SHOW CL

Unit	Line	Job	Options
CLO	6	2	[TAB,LC,LFIN,CR,CTRL]
CL1	10	none	[LC,LFOUT,LFIN,CR,CTRL]
CL2	none	none	(spooled)

SHOW COMMANDS

The SHOW COMMANDS command is used to display the currently available user-defined command definitions. See the beginning of this chapter for more information on declaring and using user-defined commands. Note that escape characters embedded in commands are represented by a dollar sign ("\$").

Example:

.SHOW COMMANDS

CLOO	==	DISPLAY \$[H\$[J
DEF	==	ASS ^ DK
EXAM	==	R KED ^/I
KPAD	==	DISPLAY \$=
NEW	==	R DIR ^/D
NOKPAD	==	DISPLAY \$>
Q	==	SPOOL LP,STAT
SUB	==	SH SUB
TOME	==	DISMO LDO_ASS DL2 DK_SH SUB
WORK	==	DISMO LDO_MOU LDO DL2:WORK DK_SH SUB

SHOW CONFIGURATION

The SHOW CONFIGURATION causes a display of certain hardware and operating system characteristics. It is equivalent to SHOW DEVICES and SHOW TERMINALS.

SHOW CORTIM

The SHOW CORTIM command displays the current value of the system parameter CORTIM. See the TSX-Plus System Manager's Guide for more information on the significance of the CORTIM parameter.

Example:

.SHOW CORTIM

2

SHOW DEVICES

The SHOW DEVICES command displays information about which devices were specified as being available when TSX-Plus was generated. For each device the following items of information are displayed:

1. Device name
2. Device status word (See the RT-11 Programmer's Reference Manual, .DSTATUS request, for a description of this value.)
3. Device handler virtual base address (load address reported by .DSTAT request; 120000 for mapped handlers)
4. Device handler physical base address (22-bit physical address divided by 100 octal; 0 for unmapped handlers)
5. Device handler size (decimal bytes)
6. Device CSR (Control and Status Register) address
7. Device interrupt vector(s) address

The CSR and vector values are only displayed if the user issuing the SHOW DEVICES command has operator privilege. Some of this information is not displayed for pseudo-devices such as TT, NL, and LD.

Example:.SHOW DEVICES

Device	Status	Handler V. base	Handler P. base	Handler size	CSR	Vector
-----	-----	-----	-----	-----	-----	-----
TT	000004					
LD	102446					
CL	006057	054234	000000	376		
DP	100021	120000	046224	446	176700	254
RK	100000	120000	046217	280	177400	220
MT	016011	105502	000000	3914	172520	224
DX	102022	120000	046205	594	177170	264
LP	020003	120000	046200	318	177514	200
NL	000025	120000	046177	58		

Keyboard Commands

SHOW HIPRCT

The SHOW HIPRCT command displays the current value of the system parameter HIPRCT. See the TSX-Plus System Manager's Guide for more information on the significance of the HIPRCT parameter.

Example:

```
.SHOW HIPRCT
40
```

SHOW INTIOC

The SHOW INTIOC command displays the current value of the system parameter INTIOC. See the TSX-Plus System Manager's Guide for more information on the significance of the INTIOC parameter.

Example:

```
.SHOW INTIOC
30
```

SHOW JOBS

The SHOW JOBS command displays information about jobs that are currently logged onto the system. The information displayed by this command is identical to that displayed by the SYSTAT command.

SHOW MEMORY

The SHOW MEMORY command displays information about memory usage including the total installed memory on the machine, the size of TSX-Plus and handlers, the memory space available to user jobs and the current job memory allocation and maximum authorized size. It also lists the size of the swappable job context area which is a system table associated with each job.

Example:.SHOW MEMORY

Total installed memory = 1280Kb
 Size of unmapped TSX and handlers = 38Kb
 Size of mapped TSX system regions = 38Kb
 Total size of TSX and mapped data = 80Kb
 Size of sharable run-time systems = 0Kb
 Size of data cache buffer area = 516Kb
 Space available for user jobs = 683Kb
 Swappable context area for each job = 4Kb
 Current job memory limit = 56Kb
 Maximum job memory limit = 64Kb

SHOW MOUNTS

The SHOW MOUNTS command displays the names of those devices that have been mounted by use of the MOUNT command and whose directories are being cached. TSX-Plus maintains a list of which jobs have mounted each device. This list is used with the INITIALIZE and SQUEEZE commands to reduce the chance of data destruction by altering a device directory while another job has a file open on the device. For logical subset disks, the name of the file which contains the device is also shown. Note that with nested logical subset disks, the table formatting is relaxed.

Example:.SHOW MOUNTS

Device	Associated jobs							
	1	2	6	8	9	10	11	
DL0:	1	2	6	8	9	10	11	
DL1:	1	6	9					
DL2:SMADA	8							
DL3:	2	11						
DL3:WORK	11							
DL3:WORK:TEMP	11							
DL3:MANUAL	2							

SHOW NUMDC

The SHOW NUMDC command displays the current value of the system parameter NUMDC. See the TSX-Plus System Manager's Guide for more information on the significance of the NUMDC parameter.

Example:.SHOW NUMDC

0

Keyboard Commands

SHOW PRIORITY

The SHOW PRIORITY command displays the current and maximum priority values for a job. In addition, the range of priority values which are used for high and low fixed priority jobs is shown. See Appendix H for further information about job execution priorities.

Example:

.SHOW PRIORITY

Current priority = 50; maximum authorized priority = 127

Low priority range = 0 to 19

High priority range = 80 to 127

SHOW QUANxx

The SHOW QUANxx command is used to display the current value of various system tuning parameters. The parameters which may be shown are: QUANO, QUAN1, QUAN1A, QUAN1B, QUAN1C, QUAN2, and QUAN3. Related tuning parameters which may also be shown are: CACHE, CORTIM, HIPRCT, INTIOC, and NUMDC. See the TSX-Plus System Manager's Guide for more information on the significance of the various system tuning parameters.

Example:

.SHOW QUAN3

20

SHOW QUEUE

The SHOW QUEUE command displays information about print files in the spool queue. The following information is displayed for each print file in the queue: the id-number assigned by the system to each file; the name of the device for which the file is queued; an asterisk if the file is currently being printed; the name of the file -- if a file name was specified with the .ENTER -- otherwise the name of the program that created the file; the name of the form on which the file is to be printed; the number of blocks in the file remaining to be printed -- this will decrease as the file is printed. For more information on printer spooling, see Chapter 5.

Example:

.SHOW QUEUE

ID	Dev	Job	File	Form	Blocks
12	LP *	1	PLT50	STD	244
13	LP	5	EXTERN	STD	5
15	LP	5	PAYROL	FORM3	35

SHOW RUN-TIMES

The SHOW RUN-TIMES command causes the display of the names of the shared run-time systems that were loaded with the system.

Example:

```
.SHOW RUN-TIMES
CBRTS
RTCOM
```

SHOW SPOOL

The SHOW SPOOL command lists the devices which have been declared as spooled during system generation. See Chapter 5 for more information on device spooling.

Example:

```
.SHOW SPOOL
Spooled devices: LP CL2
```

SHOW SUBSETS

The SHOW SUBSETS command is used to obtain information about logical subset disks which have been mounted. Only the logical subset disks which have been mounted by the user are shown; those mounted by other users are not included. The name of the logical subset disk, the file with which it is associated, the file size, and an optional asterisk are displayed. If present, the asterisk indicates that the file associated with the logical subset disk is missing. In order to DISMOUNT a logical subset disk which is associated with a missing file, it is necessary to create or copy the missing file to the appropriate device. See the MOUNT and DISMOUNT commands and Appendix G for more information on logical subset disks.

Example:

```
.SHOW SUBSETS
LD0 --> DL1:MANUAL.DSK[2601]
LD2 --> LD0:SUB.DSK[100] *
```

SHOW TERMINALS

The SHOW TERMINALS command gives a display of the parameters of all primary time-sharing terminal lines defined to TSX-Plus during system generation along with some current characteristics.

Keyboard Commands

Example:

.SHOW TERM

Line	Type	Vector	CSR	Terminal	Speed	Active	User
1	Operator	DL 060	177560	VT100	9600	Yes	CONNOR
2*	Local	DH - 0 330	160060	VT100	9600	Yes	RETTEN
3	Local	DH - 1 330	160060	VT100	9600	No	
4	Phone	DH - 2 330	160060	VT100	Auto	No	
5	Local	DH - 3 330	160060	LA120	Auto	No	
6	Local	DH - 4 330	160060	VT100	9600	No	CL unit 0
7	Local	DH - 5 330	160060	VT100	9600	Yes	BENOIT
8	Local	DH - 7 330	160060	VT200	Auto	No	
9	Local	DZ - 6 300	160040	VT100	9600	No	

The line number indicates the number assigned to each line (determined by the order of declaration during system generation). The line which issued the SHOW TERMINALS command is marked with an asterisk. The type of line may be:

Operator - the line which was declared to be the operator's console during TSX-Plus system generation

Phone - a line which has been declared to use modem control

CL line - a line which was declared during system generation to be a dedicated CL line

Local - all other lines

The Vector column identifies the type of interface card to which the line is assigned. DL11 and DLV11 type interfaces are identified as DL, and will only be followed by the vector address for the line. DZ11 and DZV11 type multiplexers will be identified as DZ and followed by the port number assigned to the line as well as the common interrupt vector for the multiplexer. DH11 type multiplexers are identified as DH, DHV11 type multiplexers are identified as DHV, and both are otherwise similar to the information provided for DZ. Three special types are provided to identify the ports on a PRO-350: PC - Professional Console; PP - Professional Printer Port; CP - Professional Communications Port. The Vector and CSR addresses are those declared during system generation. The Terminal identification is that which has been declared to TSX-Plus either during system generation or via a later SET TT <type> command. Note that only the type currently specified for the primary line is shown here. If a SET TT <type> command is issued from a virtual line, that setting does not affect the primary line and is not shown by the SHOW TERMINALS command. The terminal type setting for a virtual line may be identified from the SYSMON utility. The Speed column indicates the baud rate specified for lines for which the baud rate is known. The speed should always be known for multiplexer lines. The speed for DL lines is undefined unless the speed has been declared during system generation or a later SET TT n SPEED=speed command.

Keyboard Commands

Lines which are currently logged on are marked with a "Yes" in the Active column and the owner is identified in the User column if possible. If a time-sharing line has been "taken over" by a CL unit, then the CL unit currently attached to the line is identified. See also the SHOW CL command.

SHOW USE

The SHOW USE command causes display of the accumulated connect time and CPU usage for the current job since logon. The SHOW USE command is equivalent to the USE command.

Example:

.SHOW USE

Connect=00:50:00 CPU=00:00:23

The SPOOL Command

The SPOOL Command is used to control the operation of the spooling system. It may only be used if device spooling is enabled when the system is generated. The form of the SPOOL command is:

SPOOL device,function,parameter

where "device" is the name of a device that was specified to be spooled when the system was generated, "function" denotes what function is to be performed, and "parameter" provides additional information for some functions. A logical device name may be used in lieu of the physical device name. The available functions are:

Function	Meaning
-----	-----
ALIGN	Print form alignment file
BACK	Skip backward and resume printing
DELETE	Delete current or pending file from queue
FORM	Mount form name
[NO]HOLD	[Do not] Wait for file closure before printing
LOCK	Mount and lock form name
MULT	Print any file available for current form
SING	Request form mount for every file
SKIP	Skip forward and resume printing
STAT	Display status of spooled device

The SPOOL,DELETE command takes a slightly different form:

SPOOL device,DELETE [id-number]

Keyboard Commands

which is used to delete files from the queue of files to be printed. Note that there is a space between "DELETE" and the id-number. When no "id-number" is specified, then the file currently being printed on "device" is deleted. When "id-number" is specified, then the file associated with that "id-number" is deleted and "device" may be any spooled device. The id-number assigned to a file may be determined from the SHOW QUEUE or SPOOL device, STAT commands.

See Chapter 5 for further information on the SPOOL command.

The SQUEEZE Command

The TSX-Plus SQUEEZE command has the same form and options as the RT-11 SQUEEZE command. However, the system device (booted device when TSX-Plus is started) may not be squeezed while running TSX-Plus. If it is necessary to squeeze the system device, it must be done from RT-11. The following error message appears:

?KMON-F-This operation not legal with SY (system) device

A device which has been allocated by another user cannot be squeezed. An error message will result, similar to:

?KMON-F-Device is allocated to job 13

A device cannot be squeezed if any other user has MOUNTed the device. Attempts to SQUEEZE a device which is mounted by another user will result in the error message:

?KMON-F-Device is mounted by another user

The SHOW MOUNTS command may be used to determine which other jobs have the device mounted. If the device is mounted only by the job which squeezes it, then the directory and data caches are cleared after the operation and caching is resumed.

If terminal logging is being done (see the SET LOG command) and a log file is open on the device being squeezed, the log file is automatically closed and the following warning message appears:

?KMON-W-Closing log file

Warning: It is still possible to write to a device without first mounting it. If one user has a file open on a device while another user squeezes the device, the possibility of severe data corruption exists. If the user with the open file writes to the disk during or after the squeeze operation, the data will be directed to locations on the disk according to the previous directory information. When the disk is squeezed, these locations are no longer appropriate and write operations will probably corrupt other files. Never squeeze a disk to which another user may be writing data. As a defensive measure, always MOUNT any device which you plan to use.

The SYSTAT Command

The SYSTAT (SYstem STATus) command displays information about the performance of the system and information about each logged-on job. The SHOW JOBS and WHO commands are synonyms for the SYSTAT command. The first line indicates the amount of time since TSX-Plus was started or the RESET command was issued. The second line of the system performance information shows a breakdown of the percent of total time spent running user jobs, waiting for user I/O, waiting for swapping I/O and waiting for something to do (idle time). These percentages should add up to approximately 100% (less rounding errors). The third line of system performance information shows the percent of the total time that some user I/O and swapping I/O was being performed. The percentages on this line are not expected to sum to 100 because they simply indicate how much of the time some I/O was taking place without considering whether jobs were running while the I/O was going on. The difference between the I/O percentages on the second line and the I/O wait percentages on the first line is a measure of the amount of overlap of job execution with I/O that took place. The RESET keyboard command may be used to reset these statistics so that the statistics may be gathered over a desired interval of system execution.

The information displayed for running jobs consists of one line per job. The first item is the job number, followed by an asterisk for the line to which you are currently attached. The next item indicates the primary line and virtual line number for each job. If the job is a primary line, then the line number will correspond to the job number and the number in parentheses will be zero - indicating relative virtual job number 0. If the job is a virtual job, the line number will indicate the primary line from which it was started and the virtual job number relative to the primary line in parentheses. Detached jobs will be marked by "Det.". The next item is the current priority for each job. A two-character state code is printed next, indicating the current state of the job. The state codes and their meanings are as follows:

State	Meaning
-----	-----
HI	High priority non-interactive run state
IN	Interactive state
IO	Waiting for non-terminal I/O to finish
LO	Fixed-low-priority run state
MI	Waiting for mapped I/O buffer
MS	Waiting for a message
RN	Normal priority non-interactive run state
RT	Fixed-high-priority run state
SF	Waiting for access to a shared file
SL	Doing a timed wait (.TWAIT) or .SPND
SP	Waiting for free spool block or file entry
TI	Waiting for input from the terminal
TO	Waiting for the terminal to print output
US	Waiting for access to file management module

Keyboard Commands

The characters "-Swap" are printed following the state code if the job is currently swapped out of memory. "-Lock" is displayed by the state code if the job has locked itself in memory. The number of K-bytes of memory space used by the job, shown next, includes ordinary job space, extended memory regions and the job context block (currently 4Kb). This is followed by the connect time and CPU time used so far by the job, and the name of the program being run on the line. Except for detached jobs, which are not associated with any particular user or line, the last item indicates the user name. The user name may be blank.

Example:

.SYSTAT

Uptime: 05:34:29

System use: Run=10%, I/O-wait=8%, Swap-wait=0%, Idle=81%

I/O Activity: User I/O=10%, Swapping I/O=0%

Job	Line	Pri	State	Size	Connect	CPU time	Program	User name
1	1(0)	50	TI-Swap	33Kb	05:35:00	00:03:40	KMON	SYSMGR
2	2(0)	40	TI	32Kb	01:53:00	00:01:36	KED	ANDY
5	5(0)	84	SL-Lock	16Kb	01:00:00	00:03:17	LAB3	VOLPE
8	8(0)	50	RN	34Kb	05:24:00	00:00:33	COBOL	MITCHELL
10	Det.	50	MS-Swap	60Kb	05:35:00	00:00:00	RTSORT	
11*	2(1)	50	IN	33Kb	01:53:00	00:01:00	KMON	ANDY

The TECO Command

The TECO command is used to initiate an editing session with the TECO editor. The TSX-Plus TECO command is equivalent to the RT-11 TECO command.

The TIME Command

The TSX-Plus TIME command has the same form and options as the RT-11 TIME command. Operator command privilege is required to set the time.

The TYPE Command

The TSX-Plus TYPE command has the same form and options as the RT-11 TYPE command.

The UCL Command

The UCL command is equivalent to the SHOW COMMANDS command.

The UNPROTECT Command

The UNPROTECT command is used to clear file protection status. The TSX-Plus UNPROTECT command has the same form and options as the RT-11 UNPROTECT command.

The USE Command

The USE command displays the accumulated connect time and CPU usage since log-on.

Example:

```
.USE
Connect=02:25:00 CPU=00:02:19
```

The WHO Command

The WHO command is used to display information about the system status and logged-on jobs. The WHO command is equivalent to the SYSTAT command.

The \$STOP Command

The \$STOP command is used to halt the execution of TSX-Plus. On some hardware configurations, this may also reboot RT-11. The form of the command is:

```
$STOP
```

The \$STOP command forces an immediate logoff of all users before stopping TSX-Plus. Operator privilege is required to use this command. If other users are logged on, the system will request verification before stopping. Any response beginning with "Y" or "y" will halt TSX-Plus; any other response will return to the monitor. For example:

```
.$STOP
Other users are logged on.
Are you sure you want to stop the system?Y
Connect=01:14:22 CPU=00:11:15
```

The \$SHUTDOWN Command

The \$SHUTDOWN command is similar to the \$STOP command in that it is used to stop TSX-Plus and may return control to RT-11. However, it differs from \$STOP in the manner in which it stops TSX-Plus. \$STOP forces the immediate logoff of all users, \$SHUTDOWN does not. Rather, it sets a flag which prevents any new users from logging on and then waits for all logged on users to log off. When

Keyboard Commands

the last user logs off, TSX-Plus is stopped and control may return to RT-11. The form of this command is:

`$SHUTDOWN`

Operator privilege is required to use this command.

2.5 RT-11 Commands not supported by TSX-Plus

The following RT-11 keyboard commands are not supported by TSX-Plus and are ignored if issued: CLOSE, GT, INSTALL, LOAD, REMOVE, RESUME, SUSPEND, UNLOAD.

The following RT-11 keyboard commands are not supported by TSX-Plus and will result in the "Unrecognizable command" error message: ABORT, B, D, E, FRUN, GET, REENTER, SAVE, SRUN, START.

The following keyboard commands have different effects under RT-11 and TSX-Plus: BOOT, RESET.

The following switches are not supported by TSX-Plus with the DIBOL, COMPILE/DIBOL or EXECUTE/DIBOL commands: /BUFFERING, /LOG, /PAGE, or /TABLES.

The following SET options are not supported by TSX-Plus: EL, EXIT, or USR. The following SET TT options are not supported by TSX-Plus: CONSOL, [NO]CRLF, [NO]FB, or WIDTH.

3. COMMAND FILES

Certain series of keyboard commands may be executed together regularly, or may be so complex that it is desirable to edit and proofread them before execution. These needs are met by the facility known as indirect command files. These are files containing a set of keyboard commands to be executed together. Command files are invoked by providing the file name preceded by an "@" symbol in response to the keyboard monitor prompt. Command files may also be invoked without the "@" if their names do not conflict with system commands. See the description of keyboard command interpretation in Chapter 2 for more information on how command files may be initiated.

TSX-Plus allows the specification of parameter strings to be inserted in a command file. The parameters are stored by TSX-Plus and inserted in the text of the command file at selected points as the command file is processed. TSX-Plus also allows program data as well as system commands to be placed in a command file. Under TSX-Plus it is possible to set up a command file so that any request for data from device "TT" comes from the command file. This allows EDIT and TECO (but not KED) commands to be placed in a command file.

Command files may be executed either using normal TSX-Plus interpretation or under the control of the IND utility. Either type of control may be specified either implicitly or explicitly. The keyboard command SET KMON [NO]IND selects the desired implicit control of command files. If KMON is set IND, then command files will normally execute under the control of the IND program. If KMON is set NOIND, then command files will normally execute under TSX-Plus control. Regardless of the setting of KMON, a file may be executed under control of IND by typing:

IND file

Regardless of the setting of KMON, a file may be started as a normal command file (not under the control of IND) by typing:

\$_ file

Examples:

1. Execute the command file DK:DAILY.COM under control of the IND utility regardless of the setting of KMON:

.IND DAILY

2. Execute the command file DL1:WEEKLY.COM under normal TSX-Plus control regardless of the setting of KMON:

.\$@DL1:WEEKLY

Command Files

3.1 Invoking command files

A command file is invoked by typing:

```
@filename [param1 [param2 ... [param6]]]]]
```

where "filename" is the name of the command file and "param1", "param2", etc. are optional parameter string arguments to the command file. The default extension for a command file is "COM".

When the name of a command file does not conflict with a system command, the command file may be invoked by typing:

```
filename [param1,...param6]
```

That is, the "@" may be left out. With or without the "@" sign, if no device is specified device "DK" is first searched for the command file. Then, only without the "@" sign, if the named file is not found, then TSX-Plus attempts to locate the command file on device "SY". See Chapter 2 for a description of the steps used by TSX-Plus to interpret keyboard commands. Listing control for command files differs whether the "@" sign is used or not. If the "@" is used, then listing is controlled by the current value of the SET TT [NO]QUIET option. If no "@" is used, then listing is disabled -- as if a SET TT QUIET had been issued. In either case, the command file itself can control its own listing status with a SET TT [NO]QUIET command or with the control characters described later in this chapter.

3.2 Parameter strings

Parameter strings are normally delimited by spaces. Thus in the command:

```
@TSTRUN ABC 123.45 2/3
```

the string "ABC" is parameter 1, "123.45" is parameter 2 and "2/3" is parameter 3. In some cases it may be desirable to include spaces as part of a parameter string. If this is to be done, the first parameter must begin with the character "\" (backslash), and the backslash must be used as the parameter delimiter rather than spaces. For example, in the command line:

```
@TSTRUN \A STRING\OF PEARLS
```

parameter 1 is "A STRING" and parameter 2 is "OF PEARLS".

Up to six parameter strings may be specified when the command file is started. The total number of characters in the combined parameter strings may not exceed 60,

Parameter strings are inserted into the command file during execution at locations specified by an up-arrow ("^"). The parameter to be inserted is designated by a digit (1 to 6) immediately following the "^". During execution of the command file, the "up-arrow-digit" sequence is replaced by the parameter string passed to the command file in the position specified by the digit. If a parameter string is called for that was not specified when the command file was invoked, the up-arrow-digit will be ignored and no characters will be inserted in their place. For example, consider the following command file named TEST.COM:

```
R ^1
^3=^2
```

When this command file is invoked with the following command:

```
@TEST FORTRAN PROG
```

the command file is executed as if it contained the commands:

```
R FORTRAN
=PROG
```

Command files may be nested. That is, one command file may call another command file. When such a call occurs, the parameter strings for the outer level (calling) command file are stored on a stack in TSX-Plus and then the parameter strings for the called command file are set up. When the called command file finishes, the parameters for the calling command file are restored. The maximum depth of nesting is governed by the size of the parameter string stack and the length of the actual parameter strings. A nesting depth of 3 can be reached even with very long parameter strings. If no parameters are specified, the nesting depth may go to about 7 levels.

The command file listing status (SET TT [NO]QUIET) is also stacked as command files are nested. This means that an inner nested command file may have its listing turned on or off. When its execution is completed and control returns to the next outer level command file, the listing control is restored to the state in effect when the inner command file was called. This is not done for the outer-most command file, however, so the SET TT [NO]QUIET state in effect on completion of the outer level command file remains in effect when control is returned to the keyboard monitor.

3.3 Comments in command files

Comments may be included in command files following an exclamation point ("!"). Anything on a line in a command file following an exclamation point will be treated as comment. For example, in a command file containing the lines:

Command Files

DISPLAY Watch out! Files may be destroyed.
PAUSE Hit RETURN to proceed !Requires operator response

the resulting output would be:

Watch out
PAUSE Hit RETURN to proceed
>>

When an exclamation mark ("!") is used in a command file line to be input as data to a program, then the exclamation mark and all text following it on the input line are not passed to the program.

3.4 Command file control characters

Several combinations of characters take on special meaning when they are found within a command file. The up-arrow character ("^") followed by a letter is replaced by the control character corresponding to the letter specified. Thus "^C" becomes control-C. The escape character may be represented by up-arrow, dollar sign ("^\$"). The up-arrow character itself may be represented in the command file by preceding it with another up-arrow ("^^").

Several character combinations act as special commands to influence the execution of the command file. These commands take effect at the point of their occurrence in the command file. The characters in these special sequences are not passed to programs.

Command Sequence	Function
^(Stop listing command file. This has the same effect as SET TT QUIET, except that it only applies to the current command file.
^)	Start listing command file. This has the same effect as SET TT NOQUIET except that it only applies to the current command file.
^!	Suppress all terminal output. Both the command file listing and any program terminal output are suppressed. The "^(" and "^)" command sequences restart program generated output.
^>	Accept all terminal input from the command file. Command file data is normally only passed to programs which request lines of input (using the EMT calls: .GTLIN, .CSISPC and .CSIGEN). RT-11 handles command file data in this manner. Programs called by a command file which request terminal input with the EMT calls .TTYIN or .READ normally still expect to get this from the terminal and cannot use command file data as input to these requests. The "^>" command sequence causes <u>all</u> subsequent requests for terminal input to come from the <u>command file</u> regardless of which EMT is used. This allows data for appli-

cation programs to be placed in a command file. It also allows commands for TECO and EDIT to be placed in a command file. This command only affects input requests that occur after TSX-Plus reads the "^>" sequence. See also the description in Chapter 6 of the "N" and "O" program controlled terminal options that affect command file input.

^< Return to standard data mode. Subsequent command file data will only be passed to programs which do .GTLIN, .CSISPC or .CSIGEN EMTs (standard RT-11 mode).

^Z Control-Z is translated to control-C in command files.

3.5 PAUSE Command

The PAUSE command is provided to suspend the execution of a command file while the operator performs some manual operation. The form of the PAUSE command is:

PAUSE comments

where "comments" may be any string of characters. When a PAUSE command is executed, the PAUSE command is displayed at the terminal followed by ">>>". Execution of the command file is then suspended until carriage return is typed.

3.6 DISPLAY Command

The DISPLAY command causes a line of text to be displayed at the terminal. The form of the DISPLAY command is:

DISPLAY comments

where "comments" may be any line of text to be printed at the terminal. This is useful to mark progress through command files running in "quiet" mode (not being listed).

4. VIRTUAL TIME-SHARING LINES AND DETACHED JOBS

4.1 Virtual Lines

TSX-Plus provides a facility known as "virtual lines" which allows one time-sharing user to simultaneously control several running programs from a single terminal. When initially logging on to TSX-Plus, the user is connected to the "primary" time-sharing line, also called virtual line number 0 (zero). At any time, the user may switch to a different virtual line. This has the effect of logically disconnecting the time-sharing terminal from the current virtual line and connecting it to a different logical line. Note that when normal priority jobs are disconnected from a terminal by switching to a virtual line, the priority of the disconnected job is reduced by an amount selected during system generation. Jobs in either the fixed-high- or fixed-low-priority ranges do not suffer this priority reduction.

If a program is running when the switch is made, its execution is not affected (but it is given a lower CPU priority). If a program is running on a line that is not currently connected to the terminal and that program writes output to the terminal, the output is not displayed at the terminal, but is instead stored in a terminal output buffer. When this buffer is filled, the program is suspended (and the job may be swapped out of memory) until the terminal is reconnected to the virtual line. On a disconnected virtual line, if the output buffer becomes full or if a program requests terminal input, the bell is rung on the line which is currently connected to the terminal to indicate that a virtual line needs service.

To switch to a virtual line, type control-W (hold down CTRL key and press W) followed by a single digit (do not hold down CTRL while typing the digit). The digit identifies which virtual line the user wishes to access. Other users may be using virtual lines of the same relative number without conflict. The CTRL-W digit sequence may be entered at any time - even in the middle of a line of input. The command takes effect immediately and leaves the old line in an undisturbed state. (On later return to the primary line, CTRL-R may be used to display a partial input line which had already been typed.) TSX-Plus responds to CTRL-W, digit by printing "n>" on the terminal, where "n" is the relative number of the virtual line that has just been accessed. Note that the relative virtual line number does not correspond to the job number. The SYSTAT command may be used to determine the job number and, for virtual lines, to determine the primary line from which virtual lines were started and their virtual line numbers relative to the primary line. The primary line can always be accessed with the sequence "<CTRL-W>0", since the primary line is virtual line number 0. (Note that the system manager may select a signal character other than control-W to indicate a request to switch to a virtual line.)

If a start-up command file was specified for the primary line, it will also be executed when the virtual line is initiated. If the LOGON program is run as part of the start-up command file for a virtual line, it will automatically log the user on to the account specified when logging onto the primary line. Therefore, logical assignments made in a start-up or LOGON command file are defined for the virtual line. However, other logical assignments are not inherited by the virtual line. Any user defined commands in effect for the primary line are inherited by a virtual line when it is started.

Virtual Lines & Detached Jobs

The user logs off a virtual line by typing the "OFF" command. When a user logs off a virtual line other than the primary line, that virtual line is returned to the pool of free lines and the user is automatically reconnected to the primary line. When a user logs off the primary line, not only is the primary line released, but any virtual lines the user may have been using are also released, and the user is disconnected from the system. Note that it is possible (and frequently desirable) to switch back and forth between the primary line and one or more virtual lines without logging off any.

The total number of virtual lines and the maximum number of virtual lines which any user may utilize at a given time are determined by the system manager. An attempt to log on to a virtual line number higher than the maximum allowed for each user, is ignored. The CTRL-W, digit sequence is also ignored if the user is already connected to the maximum number of virtual lines allowed or if no free virtual lines are available.

Virtual lines are quite useful in situations where the user wishes to execute a long "number crunching" job without tying up a line. Simply start the long job and switch to another virtual line. The CPU priority of the disconnected job is reduced so that it interferes little with normal time-sharing operations, but rather "soaks up" idle CPU time. Such low priority jobs run only when no high priority jobs are running.

The keypad editors KED and K52 use the CTRL-W key to repaint the screen. Under TSX-Plus, use two successive CTRL-W's to repaint the screen with these editors. A conflict arises when switching to a virtual line while in a keypad editor and performing another editing job on the second virtual line. When exiting the second edit job, the editor automatically restores the terminal from alternate (application) keypad mode to numeric keypad mode. Then, on returning to the first editing job, it is not possible to use keypad commands since the keys no longer generate the correct sequences. To avoid this, either do not use keypad editors on more than one virtual line simultaneously, or restore the terminal to alternate keypad mode before returning to the original editing job. To set either VT-100 or VT-52 type terminals to the alternate keypad mode, send the two character sequence "<ESC>=" to the terminal.

4.2 Detached jobs

The TSX-Plus Detached job facility is similar to the virtual line facility: they both allow a timesharing user to execute several jobs simultaneously. The major difference between detached jobs and virtual lines is that you may switch back and forth among virtual lines, but once a detached job is started it is dissociated from any terminal. Detached jobs operate more like a "batch" facility. ALL terminal input for a detached job must come from a command file. Any terminal output generated by a detached job is discarded. However, terminal logging may be enabled for detached jobs to accept their terminal output; see the SET LOG command.

The differences between virtual lines and detached jobs are summarized below.

1. With virtual lines, terminal communication may be switched between several running jobs. A detached job must receive all its terminal input from a command file and any terminal output it generates is discarded (unless terminal logging directs the output to a log file). If a detached job requests terminal input and is unable to obtain it from a command file, the job is terminated.
2. Virtual lines are "owned" by a physical line. When the physical line logs off, the virtual lines also log off. Detached jobs are not associated with any physical line. Once started, any or all time-sharing users may log off without affecting detached jobs.
3. Detached jobs may be started automatically when TSX-Plus is initiated. Virtual line jobs must be started by time-sharing users after TSX-Plus is running.
4. When a detached job reaches the end of its command file and asks for more terminal input, the job is aborted and the detached job slot is freed. Virtual lines wait for more input.

There are three ways to start detached jobs: 1) during system generation specify a command file to be started as a detached job whenever TSX-Plus is started; 2) use the keyboard DETACH command; 3) use the TSX-Plus EMT to start a detached job from a running program. Detached jobs that are initiated automatically when the system is started run with operator privilege. Detached jobs initiated by use of the DETACH command or the EMT inherit the privilege of the user starting them.

4.2.1 The DETACH command.

The DETACH command is used to start a detached job, check its status and abort the job. The system manager may restrict the use of the DETACH command.

4.2.1.1 Starting a detached job: The form of the DETACH command used to start a detached job is:

DETACH file-spec

where "file-spec" is the name of a command file to be executed as a detached job. The default device is "SY:" and the default extension is "COM". The default disk "DK" is not used because no logical assignments are passed to the detached job, either in the detach command itself or in the command file started with it. That is, the file-spec must include a physical device specification and may not refer to a logical device assignment or to a file residing in a logical subset disk.

Virtual Lines & Detached Jobs

When a request is made to start a detached job, TSX-Plus searches for a free detached job slot. The total number of such job slots is established when TSX-Plus is generated. If a free slot is found, the job is started and a message is printed saying which job slot was used. This number may be used later to reference the detached job. The SYSTAT command also indicates detached job line numbers and flags detached jobs with "Det.". A detached job started by use of the DETACH command inherits the privileges (but not any logical assignments nor any logical subset disks declared after logon) of the user who is starting the job. If "file-spec" is preceded by the symbol "@", then the first line of the command file will be used by the detached job, but any remaining commands in the file will be associated with the job currently connected to the terminal.

Neither the DETACH command itself nor the command file started with it inherit any logical device assignments.

Examples:

1. Start a command file "SY:PURGE.COM" as a detached job.

```
.DETACH PURGE
Job started on line #9
```

2. Start a command file named "RK1:STATS.NEW".

```
.DETACH RK1:STATS.NEW
Job started on line #8
```

3. Start the first line of "SY:MAKIT.COM" as a detached job, and use subsequent lines as an ordinary command file.

```
.DETACH @MAKIT
Job started on line #9
```

The system manager determines the number of slots available for detached jobs and who may use detached jobs. The job numbers assigned to detached jobs start after the slots reserved for primary time-sharing lines. Virtual lines are assigned line numbers above any slots reserved for detached jobs.

4.2.1.2 Checking the status of a detached job: The form of the DETACH command used to check the status of a detached job is:

```
DETACH/CHECK line-number
```

where "line-number" is the job slot number listed when the job was started. TSX-Plus displays the status (active or free) of the detached job.

Examples:

1. Check the status of the job on line #4.

```
.DETACH/CHECK 4  
Line is active
```

2. Check the status of the job on line #5.

```
.DETACH/CHECK 5  
Line is free
```

4.2.1.3 Aborting a detached job: The form of the DETACH command used to abort a running detached job is:

```
DETACH/KILL line-number
```

where "line-number" is the job slot number listed when the job was started. For example, to kill the detached job on line #4:

```
.DETACH/KILL 4  
Job aborted
```

4.2.2 Detached job control EMTs.

TSX-Plus provides a set of EMTs that can be used to control the operation of detached jobs. Using these EMTs it is possible to start a detached job, kill a detached job and check the status of a detached job.

4.2.2.1 Starting A Detached Job: This EMT can be used to start the execution of a detached job. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,132  
.WORD    name-address
```

where "name-address" is the address of an area containing the name of the command file to be started as a detached job. The command file name must be stored in ASCIZ form and may contain an extension. If a free detached job line is available, the specified command file is initiated as a detached job and the number of the detached job line is returned in R0. A detached job started by use of this EMT inherits the privileges of the job that is executing the EMT. If there are no free detached job lines, the carry bit will be set on return.

Virtual Lines & Detached Jobs

Example:

```
.TITLE STRTDJ
.ENABL LC
```

;Start a job on a TSX-Plus detached line

```
CR      = 15
LF      = 12
ERRBYT  = 52
```

```
.MCALL .ENTER,.WRITW,.CLOSE,.PRINT,.EXIT
```

```
START: CLR      R1              ;Channel number
;No .FETCH is necessary under TSX-Plus, handlers are always resident.
1$:      .ENTER  #AREA,R1,#FILR50,#1      ;Open a one block file
                                           ;on first free channel.

        BCC      2$              ;Branch on successful .ENTER
        TSTB     @#ERRBYT         ;Why didn't .ENTER work?
        BNE      NOROOM          ;Error = 1 :not enough room for file
        INC      R1              ;Try next higher channel
        CMP      R1,#17          ;Last channel? .CDFN not supported by
                                           ;TSX-Plus, so legal channels are 0-15.
        BLE      1$              ;OK, retry on next channel
        .PRINT   #NCA            ;Ran out of channels
        BR       DONE

2$:      .WRITW   #AREA,R1,#COMNDS,#<<CMDEND-COMNDS+1>/2>,#0
        BCS      WRERR           ;Bad write?
        .CLOSE   R1             ;Close the file
        MOV      #STRTDJ,R0      ;Point to EMT arguments to
        EMT      375            ;Start the detached job
        BCS      DTCHER         ;Bad start of detached job?
        BR       DONE

NOROOM:  .PRINT   #NER           ;Not enough room error
        BR       DONE

WRERR:   .PRINT   #BADWRT        ;.WRITW error
        .CLOSE   R1
        BR       DONE

DTCHER:  .PRINT   #BADDET        ;STRTDJ error
DONE:    .EXIT

AREA:    .BLKW    5              ;EMT Argument area
STRTDJ:  .BYTE    0,132         ;EMT arguments to start a detached job
        .WORD    FILNAM        ;Pointer to name of command file
        .NLIST    BEX

FILR50:  .RAD50    /SY CKSTATCOM/ ;RAD50 name of command file to be detached
FILNAM:  .ASCIZ    /SY:CKSTAT.COM/ ;ASCII name of command file to be detached
COMNDS:  .ASCIZ    /R CKSTAT/<15><12> ;Start a monitoring program
```


CMDEND:

```

NER:      .ASCIZ  /?STRTDJ-F-Not enough room for command file./<7>
NCA:      .ASCIZ  /?STRTDJ-F-No channels available for command file./<7>
BADWRT:   .ASCIZ  /?STRTDJ-F-Error writing command file./<7>
BADDET:   .ASCIZ  /?STRTDJ-F-Error starting detached job./<7>

```

```

      .END      START

```

4.2.2.2 Aborting a detached job: This EMT may be used to abort a detached job.
The form of the EMT is:

```

      EMT      375

```

with R0 pointing to the following argument block:

```

      .BYTE    2,132
      .WORD    job-number

```

where "job-number" is the job number of the detached job to be killed. If the job number is not valid for detached jobs, the carry flag will be set on return and the EMT error byte will be set to 1.

Example:

```

      .TITLE   CKABDJ
      .ENABL   LC

```

; Check status of a detached job and abort it if running

```

      .MCALL   .EXIT,.PRINT

```

```

START:  MOV     #STATDJ,R0      ;Point to EMT arg block to
      EMT      375             ;Check status of a detached job
      BCC      1$             ;If still on, kill it
      .PRINT   #NOTON         ;Else, say it isn't active
      .EXIT

1$:     MOV     #ABRTDJ,R0      ;Point to EMT arg block to
      EMT      375             ;Abort detached job
      BCS      ABERR          ;Since we checked, should never err
      .PRINT   #KILLED        ;Say we killed it
      .EXIT

ABERR:  .PRINT   #ABERMS
      .EXIT

STATDJ: .BYTE    1,132         ;EMT arg value to check detached job status
      .WORD    9.             ;Line number of detached job to be checked
ABRTDJ: .BYTE    2,132         ;EMT arg value to abort a detached job
      .WORD    9.             ;Line number of detached job to be killed

```


Virtual Lines & Detached Jobs

```
.NLIST BEX
ABERMS: .ASCIZ  /?CKABDJ-F-Invalid detached job number/<7>
NOTON:  .ASCIZ  /?CKABDJ-I-No detached job on line #9/<7>
KILLED: .ASCIZ  /Detached job on line #9 killed./
```

```
.END START
```

4.2.2.3 Checking the status of a detached job: This EMT may be used to check the status of a detached job. The form of the EMT is:

```
EMT 375
```

with R0 pointing to the following argument block:

```
.BYTE 1,132
.WORD job-number
```

where "job-number" is the number of the detached job to be checked. If the detached job is still active the EMT returns with the carry-flag cleared. If the detached job has terminated and the detached job line is free, the EMT returns with the carry-flag set.

Example:

See the example program CKABDJ in the previous section.

5. DEVICE SPOOLING

5.1 The concept of device spooling

Device spooling is a technique that provides more efficient use of slow peripheral devices. Data sent to spooled devices is automatically redirected to a high speed disk file for temporary storage and sent to the slow device when it is ready to accept it. This allows a program generating data for the spooled device to continue processing without being slowed down by the peripheral. TSX-Plus optionally provides automatic spooling to output devices such as printers, card punches and plotters. Several devices may be spooled on a system.

When a program directs output to a spooled device, the output is diverted by TSX-Plus to a spool disk data file. An entry is made in a spool file table indicating a spool file is ready for the spooled device. When the spooled device becomes free, the spool file is copied by TSX-Plus to the device. All of the processing is automatic and the user does not have to be concerned with its operation. In fact, a user can run programs without waiting for their output to be printed. Devices to be spooled must be declared when the system is generated.

Spooled device handlers such as LP must be set to the "HANG" mode of operation to work properly with the TSX-Plus spooler. This can be done with the SET command. For example:

```
.SET LP HANG
```

Since the SET command actually stores this information permanently in the disk copy of the device handler, it need only be issued once.

5.2 Directing output to spooled devices

Output is sent to a spooled device in exactly the same way it would be directed to the device if it were not spooled. For example, if the line printer (LP) were spooled, the following commands would send a FORTRAN listing to the printer:

```
.R FORTRA  
*TEST,LP:=TEST
```

or

```
.PRINT TEST.LST
```

The name of a spooled device may be used in a MACRO .ENTER command just as a non-spooled device would.

Any number of users may simultaneously write to a spooled device without conflict. TSX-Plus separates the output from each user and prints it in an orderly fashion. A user may direct output through several I/O channels to the same or different spooled devices.

Device Spooling

5.3 Operation of the spooler

The spooling system consists of a memory resident spool file control table which contains information concerning the user's spool file and a single, system-managed spool data disk file which contains the user's intercepted output. Every time a user opens an output channel to a spooled device, a new entry is created in the spool file control table. Output records directed through separate channels to the same device have separate entries in the spool file control table. Placement in this table is governed by the processing of the user's open request for the spooled device.

The total number of spooled files that may be in existence is specified when TSX-Plus is generated. A spooled file is created when an I/O channel is opened to a spooled device; the file remains in existence until all of the output is processed by the spooled device. If a program opens a channel to a spooled device and the spool file control table already contains the maximum number of spooled files, the program is suspended until a slot becomes available in the spool file control table.

All output directed to spooled devices is stored in a common disk data file. The total file space that is available for spooled data is specified when the TSX-Plus system is generated. Space within this disk file is dynamically allocated as needed on a block by block basis. Each block contains 508 bytes of user's output data. Therefore, write requests smaller than 508 bytes will be combined into one spool data block. A write request larger than 508 bytes will be split into more than one spool data blocks. If the spool data file is totally filled, programs writing to the spooled device will be suspended until space becomes available.

Actual output to the spooled devices is processed on a sequential first-in/first-out basis. Since output is actually coming from the disk file, each I/O request to the device handler is the size of the spool data block (508 bytes). While spooled files are being printed, TSX-Plus maintains a list of the blocks most recently printed. The length of this list is specified by the number of backup blocks declared during system generation. When the list becomes full, the oldest block is deallocated and the new block is added. When the entire spooled file has been successfully written, the entire list is deallocated.

Read requests from a spooled device (.READ and .SFUN requests) bypass the spooling system and connect directly with the device handler.

5.4 The SPOOL Command

The SPOOL command is used to control the operation of the spooling system. The form of the SPOOL command is:

SPOOL device,function,parameter

Device Spooling

where "device" is the name of a spooled device, "function" indicates the operation to be performed, and "parameter" is an optional item of information used with some functions.

A second form of the SPOOL command is used to delete current or pending files in the spooler queue. The form of this command is:

SPOOL device,DELETE [id-number]

where the optional id-number is assigned by the system and can be determined from the SHOW QUEUE or SPOOL device,STAT commands.

The device name may refer to either the physical name of a spooled device (such as LP, LS, or CL2) or a logical name assigned to a spooled device. For example:

.ASSIGN CL2 LP

The two following commands are then equivalent:

.SPOOL CL2,STAT
.SPOOL LP,STAT

A device specified without a unit number is equivalent to unit 0 (e.g. CL == CL0).

The FORM and LOCK Functions

The FORM and LOCK functions are used to specify the name of the currently mounted form. The form name is specified in the "parameter" field of the command. The FORM function allows TSX-Plus to request a form mount when a different form is needed. The LOCK function specifies that the form is to be locked on the printer and disables form mount request messages.

Examples:

.SPOOL LP,FORM,BILLS
.SPOOL LP,LOCK,BILLS
.SPOOL LP,FORM,STD

Note the difference between the FORM keyboard command and the SPOOL command with the FORM or LOCK functions. The FORM command is used by the TSX-Plus user to specify the default form name to be used for subsequent spool files. The SPOOL FORM/LOCK commands are used by the TSX-Plus operator to tell the spooling system which form is currently mounted.

The ALIGN Function

The ALIGN function is used to print a form alignment file on a spooled device. The name of the alignment file is specified in the "parameter" field of the command. The default file extension for form alignment files is "ALN".

Device Spooling

Examples:

```
.SPOOL LP,ALIGN,BILLS
.SPOOL LP,ALIGN,RK1:PAYROL.DAT
.SPOOL LP,ALIGN,DX:RPORT2
```

The DELETE Function

The DELETE function is used to remove a file from the list of files to be printed by the spooler. This form of the SPOOL command is:

```
SPOOL device,DELETE [id-number]
```

If the id-number is omitted, then the file currently being printed on the specified device is aborted. If the file is open, the file is marked to be deleted but is not actually deleted until it is closed and the spooler begins to process it. If the file is currently being printed when the delete command is issued, the spooler deletes blocks as they become available and the spooled device is tied up until the file deletion is completed.

When an id-number is specified, that number is used to identify the file to be deleted. In this case, "device" may be the name of any spooled device and does not have to match the device on which the file is actually being printed.

A spooled file id-number may be determined from either the SHOW QUEUE command or the SPOOL device,STAT command. For example:

<u>.SHOW QUEUE</u>					
<u>ID</u>	<u>Dev</u>	<u>Job</u>	<u>File</u>	<u>Form</u>	<u>Blocks</u>
37	LP *	1	PAYROL	STD	135
39	LP	1	GL	STD	25
41	LP	1	AR	STD	86

Examples:

To delete the file "PAYROL" currently being printed on LP:,

```
.SPOOL LP,DELETE
```

To delete the file "AR" which has not yet started printing,

```
.SPOOL LP,DELETE 41
```

The SKIP Function

The SKIP function causes the spooler to skip over the next n blocks in the spool file that is currently being printed, where n is specified in the parameter field of the instruction. Each block in the spool data file contains 508 characters. Printing of the file continues after the indicated number of blocks have been skipped.

Examples:

```
.SPOOL LP,SKIP,10
.SPOOL LP,SKIP,100
```

The BACK Function

The BACK function causes the spooler to skip backward in the spool file a number of blocks and then resume printing at that point. The number of blocks involved is specified when TSX-Plus is generated. Each block in the spool data file contains 508 characters. This function is particularly useful for recovering from paper tears or remounts. The spooler will finish printing the current block before backing up.

Examples:

```
.SPOOL LP,BACK
.SPOOL LS,BACK
```

The STAT Function

The STAT function is used to determine the status of a spooled device. Information returned includes: the condition of the spooler (active, idle, or waiting for a form mount); the name of the currently mounted form; and information about files waiting to be printed on the device. The SHOW QUEUE command may also be used to display information about files in the spooler queue.

Example:

```
.SPOOL LP,STAT
Spooler idle
Currently mounted form = STD
No files in queue
```

The SING and MULT Functions

The SING and MULT functions control how the spooler will handle multiple files queued for the same form. In "MULT" mode (the initial setting) no form mount request message is generated if a spool file is found that needs the currently mounted form. Processing of the file begins automatically.

In "SING" mode a form mount request is generated for every file even if the file needs the currently mounted form. This is useful where equipment setup or form alignment is needed for every file.

Examples:

```
.SPOOL LP,SING
.SPOOL LP,MULT
```


Device Spooling

The HOLD and NOHOLD Functions

A spooled device that is in the HOLD mode will not begin to process a spool file until the file is completely created and the I/O channel associated with the file is closed. A spooled device that is in NOHOLD mode will begin to process a spool file as the file is being created. In NOHOLD mode, the spooler will begin to process a file sooner; however, if the file is being created slowly, the spooled device will remain busy (and unavailable to other users) for as long as it takes to finish generating the file. If the spool storage file is completely filled, the spooler will attempt to free space by beginning to process open spool files even if HOLD is in effect.

The default [NO]HOLD mode is established when the TSX-Plus system is generated. A [NO]HOLD command remains in effect until another [NO]HOLD command is issued or the system is restarted.

Examples:

```
.SPOOL LP,NOHOLD  
.SPOOL LP,HOLD
```

It is also possible to dynamically request that a file being printed through the spooler be either held until the file is closed or begin printing as data is made available from the program. This could be used in a situation where NOHOLD is the normal condition, but a program which uses the printer generates data slowly. If data were passed to the printer as soon as available, then printer output from all other jobs would be delayed until the slow job closes the output. This can be avoided by the having the slow program select hold mode for its output. Then, other jobs can proceed to use the printer without being delayed by the slow job. The form of the EMT to select hold or nohold mode on an individual file basis is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE   chan,151  
.WORD   0  
.WORD   flag
```

where "chan" is the channel number which has been used to open the print file and "flag" indicates whether the file is to be printed as it is generated or held until the file is closed. If "flag"=0, the output is printed as generated (equivalent to NOHOLD); if "flag"=1, then the output is not printed until the file is closed. This EMT must be issued after a channel has been opened to the printer (through the spooler), but before any data has been written to it. If the channel is open to any non-spooled device, then the EMT is ignored.

Example:

```

        .TITLE  SPHOLD
        .ENABL  LC
;
; Demonstrate the EMT to hold spooler output until the file is closed
;
        .MCALL  .CSISPC,.LOOKUP,.READW,.WRITW,.CLOSE,.EXIT
ERRBYT  = 52                                ;EMT error code byte location
START:  .CSISPC #OUTSPC,#DEFEXT,#0          ;Get name of file to copy
        BCS     START                      ;Proceed unless error
        MOV     #INSPC,R1                 ;Point to first input filspc
OPNFIL: .LOOKUP #AREA,#0,R1                ;Try to open input file
        BCS     START                      ;Get a new command on error
        MOV     #OUTSPC,R2                 ;Point to output filspc
        MOV     #^RLP ,(R2)+              ;Put LP: in output filspc
        ADD     #2,R1                     ;Point to input filspc filename
        MOV     (R1)+,(R2)+                ;Move file name into LP filspc
        MOV     (R1)+,(R2)+                ; (not necessary, but convenient)
        TST     (R1)+                      ;Skip over file extension
        .LOOKUP #AREA,#1,#OUTSPC          ;Open channel to printer (spooled)
        BCC     NOHOLD                     ;Proceed unless error
GIVEUP: .CLOSE  #0                          ;Close input file
        .EXIT                               ;And give up
;
; Tell spooler to hold file until it is closed
; (must be issued before any writes to file)
;
NOHOLD: MOV     #SPHOLD,R0                  ;Point to EMT arg block to
        EMT     375                         ;Hold output until close
        CLR     R2                          ;Initialize block pointer
6$:     .READW  #AREA,#0,#BUFFER,#256.,R2 ;Copy a block from the file
        BCS     NXTFIL                      ;Try next file on error
        .WRITW  #AREA,#1,#BUFFER,#256.,R2 ;Copy the file block to LP
        BCC     8$                          ;Error?
        .CLOSE  #1                          ;Close print file
        BR      GIVEUP                       ;Forget it
8$:     INC     R2                          ;Point to next block
        BR      6$                          ;And get next block
NXTFIL: .CLOSE  #0                          ;Close input file
        .CLOSE  #1                          ;and print file
        TST     2(R1)                       ;Any input file?
        BNE     OPNFIL                      ;Repeat if so
        BR      START                      ;Else ask for more files

AREA:   .BLKW   10                          ;General EMT arg block area
SPHOLD: .BYTE   1,151                       ;EMT arg block to hold spool output
        .WORD   0                          ;on channel 1 until file is closed
HNH:    .WORD   1                          ;HNH=0 immed; HNH=1 hold til close
OUTSPC: .BLKW   15.                         ;Output file specs

```


Device Spooling

INSPC:	.BLKW	24.	;Input file specs
DEFEXT:	.WORD	0,0,0,0	;No default file types
BUFFER:	.BLKW	256.	;I/O buffer area
	.END	START	

5.5 Use of special forms with spooled devices

Output files directed to spooled devices are queued and held until the spooled device becomes free. Because of this, a special procedure is required to synchronize the mounting of a special form with the printing of a file that requires the form.

If the first character in a file directed to a spooled device is a right square bracket ("]"), TSX-Plus will interpret the following one to six characters in the file as the name of the form on which the file should be printed. Form names may be from one to six characters in length and must be specified immediately following the initial square bracket character. The form name must be terminated with a carriage return, line feed. Square bracket characters are not significant to the spooler in any position other than the first character of the file.

If a spooled file does not begin with a right square bracket character, TSX-Plus uses the form name that was last specified by the user with a FORM command (see Chapter 2). If no FORM command has been issued by the user, TSX-Plus uses the form name "STD" for the file.

Each time TSX-Plus selects a file to be printed on a spooled device, it first looks for a waiting file that requires the form currently mounted on the spooled device. If several such files are available, the oldest one is started. If no file can be found that requires the currently mounted form, TSX-Plus selects the oldest file requiring a different form and issues a form mount request. The message appears on the the operator's terminal as:

"Mount 'XXXXXX' form on dd"

where "XXXXXX" represents the form name and "dd" is the spooled device. The terminal to which the message is directed is the one declared as the operator's console when the TSX-Plus system was generated.

Once the form mount request message is sent, the spooler for the device requiring the new form is suspended. In order to restart the spooler the operator must issue a SPOOL-FORM or SPOOL-LOCK command. These commands tell the spooler that a particular form has been mounted and is ready for use. The operator does not have to mount the form called for in the form mount request. He may mount any form he desires, in which case TSX-Plus will search for a spool file that needs the form actually mounted.

The SPOOL-FORM and SPOOL-LOCK commands are both used by the operator to indicate which form has been mounted; however, there is a difference in the effect of the two commands. After processing all files that need the currently

mounted form, TSX-Plus checks for files requiring a different form. If there are any, it checks to see if the current form was mounted using a SPOOL-FORM or SPOOL-LOCK command. If a SPOOL-FORM command was used, TSX-Plus issues a form mount request message. If a SPOOL-LOCK command was used, TSX-Plus considers the current form to be locked on the printer and does not issue a form mount message; rather, it waits for new spool files to be created that need the currently mounted form.

5.6 Form alignment procedure

When mounting a new form it is usually necessary to verify the correct positioning of the form before starting production printing on the form. The SPOOL-ALIGN command provides this facility.

The SPOOL-ALIGN command allows the TSX-Plus operator to specify a form alignment file to be printed on the indicated spooled device. Form alignment files are printed immediately without regard to the name of the currently mounted form. The SPOOL-ALIGN command may be issued repeatedly if several attempts are required to mount a form.

Alignment files are created by the user and may contain any desired information. Typically they contain a short sample output file that matches a particular form. Alignment files should not contain a form name specification. The normal sequence of operations involving a form mount is as follows:

1. TSX-Plus issues a form-mount request message and suspends the spooler.
2. The operator mounts the desired form and issues one or more SPOOL-ALIGN commands to verify its positioning.
3. Once the form is correctly positioned, the operator issues a SPOOL-FORM or SPOOL-LOCK command to tell TSX-Plus which form has been mounted.
4. TSX-Plus begins printing the oldest file that needs the currently mounted form.

The SPOOL-ALIGN command may be issued at any time, but it is typically used between a form-mount message and a SPOOL-FORM or SPOOL-LOCK command.

6. PROGRAM CONTROLLED TERMINAL OPTIONS

6.1 Terminal input/output handling

The terminal keyboard and screen provide the principal interface between a time-sharing user and the TSX-Plus operating system. TSX-Plus accepts characters from the keyboard, echoes them to the screen, and stores them in a separate buffer for each time-sharing user. Then, when a program (either a user written program, or a utility, or the operating system keyboard monitor) requests input from the terminal, characters are removed from the internal buffer and passed to the program. The low-level requests for input from a program can call for a single character (.TTYIN), or for an entire line (.GTLIN, .CSIGEN, .CSISPC), or for a whole block of characters (.READ). Since the requests for a whole line of input are most common, TSX-Plus improves overall efficiency for many users by retaining characters typed at the keyboard in an internal buffer until a special character is typed which indicates that the line of input is complete. This special character, which indicates that keyboard input is ready, is called an activation character. The standard activation characters are carriage return and line feed. In addition, several control keys will also cause immediate system response. For example, control-C is used to abort the execution of a running program. If the program is waiting for input, one control-C will cause an immediate abort. If the program is not waiting for input, it is necessary to type two control-C's to get the system's attention and abort a program.

When a program requests terminal input, TSX-Plus puts the program in a suspended state until an activation character is typed. This state, in which a program is waiting for input, but no activation character has been typed, is identified as the TI state by the SYSTAT command. When characters are typed at the terminal, TSX-Plus responds quickly and stores them in the terminal input buffer for that line, then returns to process other jobs which need its attention. Thus, the amount of time the CPU spends processing input characters is kept to a minimum, and the amount of CPU time used by a program in the TI state is also very small. Some programs, however, request single characters with the .TTYIN request. Normally, these programs are treated by TSX-Plus just as those requesting lines of input (e.g. .GTLIN requests). That is, the job is suspended, input characters are stored in the terminal input buffer, and characters are only passed to the program after an activation character is typed. If a program requests a character with a single .TTYIN, the user can type as many characters as the terminal input buffer will hold (allocated during system generation), but the program will remain suspended and no characters are passed to the program. Then, when an activation is typed, the program is restored to an active state, the first character in the input buffer is passed to the program and processing continues. If the program requested no more characters, then on exit the remainder of the input buffer, including the activation character, would be passed to the next program (usually the keyboard monitor) which would try to interpret them. This may result in an invalid command error message.

TSX-Plus allows the programmer a wide variety of ways to influence the normal input scheme outlined above. One of the most common is the use of "single character activation". With this technique, all characters are regarded as activation characters. That is, if a program requests a single character with a .TTYIN, then as soon as a character is typed and becomes available in the

Terminal Control

input buffer, it is passed to the program and the program resumes execution. The standard way to request single character activation under RT-11 is by setting bit 12 in the user's Job Status Word (JSW). However, under TSX-Plus, this is not by itself sufficient to cause single character activation. The reason is that quite a few programs designed for a single user environment use this in a way that causes constant looping back and consequently "burns up" a large amount of processor time. In a single user environment, this is of minor importance since no other jobs are trying to use the processor at the same time. Of course, in a multi-user system, this is wasteful and should be avoided. Therefore, under TSX-Plus, setting JSW bit 12 is not by itself sufficient to initiate single character activation. It is necessary BOTH to set bit 12 and to issue a special command to TSX-Plus indicating that single character activation is actually desired. This may be done either by specifying single character activation when running the program (see the /SINGLECHAR switch described under the R command), or with the SET TT SINGLE command, or with the "S" program controlled terminal option described in this chapter. Note that when a program is in "single character activation" mode, the system does not echo terminal input, it is the program's responsibility to do so.

The situation in which a program requests single characters but none are available in the input buffer also receives special treatment. The single character input request is eventually coded as EMT 340. The .TTYIN request repeats this request until a character is finally obtained, whereas the .TTINR request supposedly permits processing to continue if no character is available. In fact, however, the EMT 340 call will itself suspend the job until a character is available from the input buffer. This is referred to as "stalling" on a .TTYIN. The purpose is to avoid the unnecessary looping back to get a character. Under RT-11, if the programmer decides not to wait for a character to become available, but rather proceed with execution, it is only necessary to set bit 6 (100 octal) in the Job Status Word. Again, some programs abuse this technique and would waste the system resources in a time-sharing environment. So, TSX-Plus requires confirmation that the user is aware of the extra system load that could be caused by the constant looping back checking for a character. This may be done either as a system command (SET TT NOWAIT), or with the /SINGLECHAR switch when running the program, or within a program by using the program controlled terminal option "U". All of these methods cause TSX-Plus to honor bit 6 in the JSW. If nowait input is truly desired under TSX-Plus, it is necessary BOTH to set bit 6 in the JSW and to tell TSX-Plus to use nowait input (R[UN]/SINGLECHAR, or SET TT NOWAIT, or the "U" terminal option).

TSX-Plus allows many other ways of modifying terminal input and output for special circumstances. These are provided to allow maximum versatility in the system while still maintaining the high efficiency needed in a multi-user environment. The programmer communicates the need for special terminal handling to the system through the use of special "program controlled terminal options". These are described individually in the next section.

6.2 Program controlled terminal options

The following table lists the functions which may be modified during program execution.

Function Character	Meaning
A	Set rubout filler character.
B	Enable VT52 & VT100 escape-letter activation.
C	Disable VT52 & VT100 escape-letter activation.
D	Define new activation character.
E	Turn on character echoing.
F	Turn off character echoing.
H	Disable virtual lines.
I	Enable lower case input.
J	Disable lower case input.
K	Enable deferred character echo mode.
L	Disable deferred character echo mode.
M	Set transparency mode for output.
N	Suspend command file input.
O	Restart command file input.
P	Reset activation character.
Q	Set activation on field width.
R	Turn on high-efficiency TTY mode.
S	Turn on single-character activation mode.
T	Turn off single-character activation mode.
U	Enable no-wait TT input test.
V	Set field width limit.
W	Turn tape mode on
X	Turn tape mode off
Y	Disable echo of line-feed after carriage-return
Z	Enable echo of line-feed after carriage-return

These functions have a temporary effect in that they are automatically reset to their normal values when a program exits to the keyboard monitor. They are not reset if the program chains to another program until control is finally returned to the monitor. Some terminal options (notably high-efficiency and single-character modes) are incompatible with and override some other terminal options.

TSX-Plus provides two methods for a running program to dynamically alter some of the parameter settings relating to the user's timesharing line. The preferred method of selecting these functions is to use the TSX-Plus EMT for that purpose. This is readily available from MACRO programs and an appropriate MACRO subroutine should be linked into jobs written in other languages. The form of the EMT to select program controlled terminal options is:

EMT 375

Terminal Control

with R0 pointing to the following argument block:

```
.BYTE    0,152
.WORD    function-code
.WORD    argument-value
```

where "function-code" is the character from the table above which selects the terminal option, and "argument-value" may be a third value used only with some of the functions. An advantage of the EMT method of selecting program controlled terminal options is that they may be used even when the terminal is in high-efficiency mode.

Example:

```
.TITLE    EMTMTH
.ENABL    LC

;
; Demonstrate TSX-Plus program controlled terminal options using
; the EMT method.
;

.MCALL    .GVAL,.PRINT,.EXIT,.TTYIN
.GLOBL    PRTDEC

;
; Determine and display current lead-in char
; Default = 35, but don't count on it
;
START:    .GVAL    #AREA,#-4.        ;Determine current leadin character
          MOV      R0,R1            ;Save it
          .PRINT   #LEADIS          ;"Current lead-in is"
          MOV      R1,R0            ;Retrieve lead-in char value
          CALL     PRTDEC           ;Display it
          .PRINT   #LEADND

;
; Set rubout filler character
;
          MOV      #SETRUB,R0       ;Point to EMT arg block to
          EMT      375              ;Set rubout filler character

;
; Now demonstrate the current rubout filler character
; Back space is the default, which we changed to underline
;
          .PRINT   #TRYIT           ;"Enter some and delete them"
          MOV      #BUFFER,R1       ;Point to input buffer
1$:       .TTYIN   ;Get next char into input buffer
          CMPB     R0,#12           ;End of input (CR/LF pair)?
          BEQ      2$              ;Yes, terminate input
          CMP      R1,#BUFEND       ;Buffer overflow?
          BLO      1$              ;Get more if not

;
2$:       .PRINT   #THANKS
```



```

.EXIT

AREA:  .BLKW   10           ;General EMT arg block
SETRUB: .BYTE   0,152       ;EMT arg block to
        .WORD   'A         ;Set rubout filler character
        .WORD   '         ; to underline

TRYIT:  .ASCII  /Enter some characters at the prompt and then /
        .ASCII  /erase them with/<15><12>/DELETE or Control-U./
        .ASCII  / They should be replaced with underlines./<15><12>*/<200>

LEADIS: .ASCII  /We don't care that the current lead-in char is /<200>
LEADND: .ASCIZ  ./

THANKS: .ASCIZ  <15><12>/STOP -- Thank you./
BUFFER: .BLKB   81.
BUFEND:

.END    START

```

When it is not practical to incorporate the EMT method of selecting program controlled terminal options into a program, an alternate method using a "lead-in" character may be used. This is conveniently done by sending a sequence of characters to the terminal using the normal terminal output operations of the language. Examples are the FORTRAN TYPE, COBOL-Plus DISPLAY, BASIC PRINT, and Pascal WRITE statements. Program controlled terminal options are selected by having the running program send the lead-in character immediately followed by the function character and for some functions a third character defining the argument value for the function. TSX-Plus intercepts the lead-in character and the one or two following characters and sets the appropriate terminal option. It does not pass these intercepted characters through to the terminal. The default value for the lead-in character is the ASCII GS character (octal value 35; decimal value 29). However, the lead-in character may be redefined during system generation when the value conflicts with other uses of the system. For example, some graphics terminals use the GS character as either a command or parameter value. Programmers should not rely on the default value of the lead-in character, but may obtain the current value of the lead-in character from the .GVAL request with an offset of -4. Note that when in high-efficiency mode (set with either the RUN/HIGH switch or the "R" program controlled terminal option) output character checking is disabled and the lead-in character method of selecting program controlled terminal options is disabled; in this case, the lead-in character, function-code character, and argument value character are passed through to the terminal. When in high-efficiency mode, the EMT method of selecting program controlled terminal options is still functional. See Chapter 7 for information on turning high-efficiency terminal mode off by means of an EMT.

Example:

PROGRAM LEADIN

```

C
C Demonstrate TSX-Plus program controlled terminal options using
C the "lead-in" character method.
C

```


Terminal Control

```
        BYTE    LEADIN(2)
        INTEGER ILEAD
        EQUIVALENCE (ILEAD,LEADIN(1))

C
C Determine and display current lead-in char
C Default = 29, but don't count on it
C
        ILEAD = ISPY(-4)          !.GVAL with offset = -4.
        TYPE 820,LEADIN(1)       !Display the current lead-in char value
C
C Set rubout filler character
C
        TYPE 800,LEADIN(1),'A','_'
C
C Now demonstrate the current rubout filler character
C Back space is the default, which we changed to underline
C
        TYPE 810                !Ask for something to be erased
        ACCEPT 830              !Wait for input before exiting
C
        STOP 'Thank you.'

800    FORMAT(1H+,A1,$)
810    FORMAT(1H0,'Enter some characters at the prompt and then ',
1      'erase them with '/' DELETE or Control-U.',
1      ' They should be replaced with underlines.'/' *', $)
820    FORMAT(' The current value of the lead-in character is ',I3,'.')
830    FORMAT(40H
        )
END
```

The following paragraphs explain the uses of each of the program controlled terminal option function-codes. Any of these options may be selected by either the EMT method or by the lead-in character method.

6.2.1 "A" function--Set rubout filler character.

When a scope type terminal is being used, the normal response of TSX-Plus to a DELETE character is to echo backspace-space-backspace which replaces the last character typed with a space. TSX-Plus responds to a CTRL-U character in a similar fashion, echoing a series of backspaces and spaces. Some programs that display forms use underscores or periods to indicate the fields where the user may enter values. In this case it is desirable for TSX-Plus to echo backspace-character-backspace for DELETE and CTRL-U where "character" may be period or underscore as used in the form. The character to use as a rubout filler is specified by the argument-value with the EMT method or by the third character with the lead-in character method.

6.2.2 "B" & "C" functions--Set VT52, VT100 and VT200 escape-letter activation. VT52, VT100 and VT200 terminals are equipped with a set of special function keys marked with arrows and other symbols. When pressed, they transmit two or three character escape sequences. The "B" function tells TSX-Plus to consider

these as activation sequences. The escape character and the letter are not echoed to the terminal, but are passed to the user program. The "C" function disables this processing and causes escape to be treated as a normal character (initial setting).

6.2.3 "D" function--Define new activation character.

Under normal circumstances TSX-Plus only schedules a job for execution and passes it a line of input when an "activation" character such as carriage return is received. The "D" function provides the user with the ability to define a set of activation characters in addition to carriage return.

The new activation character is specified by the argument-value with the EMT method or by the third character with the lead-in character method. The maximum number of activation characters that a program may define is specified when the TSX-Plus system is generated.

Using this technique, any character may be defined as an activation character, including such characters as letters, DELETE, CTRL-U, and CTRL-C. When a user-defined activation character is received, it is not echoed but is placed in the user's input buffer which is then passed to the running program.

By specifying CTRL-C as an activation character, a program may lock itself to a terminal in such a fashion that the user may not break out of the program in an uncontrolled manner.

If carriage return is specified as a user activation character, neither it nor a following line feed will be echoed to the terminal. TSX-Plus will also not add a line feed to the input passed to the program.

6.2.4 "E" and "F" functions--Control character echoing.

The "E" and "F" functions are used to turn on and off character echoing. The "E" function turns it on, and the "F" function turns it off. An example of a possible use is to turn off echoing while a password is being entered.

6.2.5 "H" function--Disable virtual line use.

The "H" function disables the virtual line facility for the time-sharing line.

6.2.6 "I" and "J" functions--Control lower case input.

The "I" function allows lower case characters to be passed to the running program. The "J" function causes TSX-Plus to translate lower case letters to upper case letters. The SET TT [NO]LC keyboard command also performs these functions.

6.2.7 "K" and "L" functions--Control character echoing.

The "K" function causes TSX-Plus to enter "deferred" character echo mode. The "L" function causes TSX-Plus to enter immediate character echo mode. Any characters in the input buffer which have not been echoed when the "L" function is selected will be immediately echoed. See the description of the SET TT [NO]DEFER command for an explanation of deferred echo mode.

Terminal Control

6.2.8 "M" function--Set transparency mode of output.

If transparency mode is set, TSX-Plus will pass through each transmitted character without performing any special checking or processing. Transparency mode allows the user's program to send any 7-bit character to the terminal. Note that once transparency mode is set on, TSX-Plus will no longer recognize the lead-in character (octal 35, which means a program control function follows). The only way to turn off transparency mode is to exit to KMON.

6.2.9 "N" and "O" Functions--Control command file input.

When a command file is being used to run programs (see Chapter 3), input which would normally come from the user's terminal is instead drawn from the command file. Occasionally, it is desirable to allow a program running from a command file to accept input from the user's terminal rather than the command file. The "N" function suspends input from the command file so that subsequent input operations will be diverted to the terminal. The "O" function redirects input to the command file. These functions are ignored by TSX-Plus if the program is not being run from a command file.

6.2.10 "P" function--Reset activation character.

The "P" function performs the complement operation to the "D" function. The "P" function is used to remove an activation character that was previously defined by the "D" function. The character to be removed from the activation character list is defined by the argument-value with the EMT method or by the third character with the lead-in character method.

Only activation characters that were previously defined by the "D" function may be removed by the "P" function.

6.2.11 "Q" function--Set activation on field width.

The "Q" function allows the user to define the width of an input field so that activation will occur if the user types in as many characters as the field width, even if no activation character is entered. The field width is specified by the ASCII code value of the argument-value with the EMT method or of the third character with the lead-in character method. If an activation character is entered before the field is filled, the program will be activated as usual. Each time activation occurs the field width is reset and must be set again for the next field by reissuing the "Q" function. For example, the following sequence of characters could be sent to TSX-Plus to establish a field width of 43 characters: "<lead-in>Q+". Note that the character "+" has the ASCII code of 053 (octal) which is 43 decimal.

6.2.12 "R" function--Turn on high-efficiency terminal mode.

The "R" function causes TSX-Plus to place the line in "high efficiency" terminal mode. The effect of this is to disable most of the character testing overhead that is done by TSX-Plus as characters are transmitted and received by the line. Before entering high-efficiency mode the program must declare a user-defined activation character that will signal the end of an input record. Once a program has entered high-efficiency mode, characters sent to the terminal are processed with minimum system overhead. For example, tab characters are not expanded to spaces. Also, TSX-Plus does not check to see if

the character being sent is the TSX-Plus terminal control "leadin" character. This means that the lead-in character method may not be used to control terminal options until the program exits or the EMT to turn off high efficiency mode is used (see Chapter 7). Characters received from the terminal are passed to the program with minimum processing: they are not echoed; and control characters such as DELETE, control-U, control-C, control-W and carriage-return are all treated as ordinary characters and passed directly to the program. High-efficiency mode terminal I/O is designed to facilitate machine-to-machine communication; it is also useful for dealing with buffered terminals that transmit a page of information at a time.

6.2.13 "S" function--Turn on single-character activation mode.

The "S" function causes TSX-Plus to allow a program to do single-character activation by setting bit 12 in the Job Status Word. Normally TSX-Plus stores characters received from the terminal and only activates the program and passes the characters to it when an activation character, such as carriage-return, is received. It does this even if bit 12 is set in the Job Status Word, which under RT-11 causes the program to be passed characters one-by-one as they are received from the terminal. The "S" function can be used to cause TSX-Plus to honor bit 12 in the Job Status Word. If JSW bit 12 is set and the program is in single-character activation mode, TSX-Plus passes characters one-by-one to the program as they are received and does not echo the characters to the terminal. The /SINGLECHAR switch for the R[UN] command and the SET TT SINGLE command can also be used to cause TSX-Plus to honor JSW bit 12. Since the high-efficiency mode implies certain terminal characteristics (such as buffered input and no echo), it is not possible to override these inherent modes by using other function codes.

6.2.14 "T" function--Turn off single-character activation mode.

The "T" function is the complement of the "S" function. It turns off single-character activation mode.

6.2.15 "U" function--Enable non-wait TT input testing.

The "U" function causes TSX-Plus to allow a program to do a .TTINR EMT that will return with the carry bit set if no terminal input is pending. Normally TSX-Plus suspends the execution of a program if it attempts to obtain a terminal character by doing a .TTINR EMT and no input characters are available. It does this even if bit 6 of the Job Status Word is set, which under RT-11 would enable non-blocking .TTINR's. This is done to prevent programs from burning up CPU time by constantly looping back to see if terminal input is available. The "U" function causes TSX-Plus to honor bit 6 in the Job Status Word and allows a program to do a .TTINR to check for pending TT input without blocking if none is available. The SET TT NOWAIT command and the /SINGLECHAR switch for the R[UN] command also perform this function. Because the single character terminal option determines several terminal operating modes (such as no echo and transparent input), it is incompatible with other terminal functions which would conflict with the implied single-character operation. See the description of special terminal mode in the RT-11 Programmer's Reference Manual.

Terminal Control

6.2.16 "V" function--Set field width limit.

The "V" function is used to set a limit on the number of characters that can be entered in the next terminal input field. Once the "V" function is used to set a field limit, if the user types in more characters to the field than the specified limit, the excess characters are discarded and the bell is rung rather than echoing the characters. An activation character still must be entered to complete the input. The field width is specified by the ASCII code value of the argument-value with the EMT method or of the third character with the lead-in character method. The field size limit is automatically reset after each field is accepted and must be re-specified for each field to which a limit is to be applied. Note the difference between the "Q" and "V" functions. The "Q" function sets a field size which causes automatic activation when the field is filled; the "V" function sets a field size which causes characters to be discarded if they exceed the field size.

6.2.17 "W" and "X" functions--Control tape mode.

The "W" function turns on "tape" mode and the "X" function turns it off. Turning on "tape" mode causes the system to ignore line-feed characters received from the terminal or external device. The SET TT [NO]TAPE keyboard command may also be used to control tape mode.

6.2.18 "Y" and "Z" functions--Control line-feed echo.

The "Y" function is used to disable the echoing of a line-feed character when a carriage-return is received. Normally, when TSX-Plus receives a carriage-return character, it echoes carriage-return and line-feed characters to the terminal and passes carriage-return and line-feed characters to the program. The "Y" function alters this behavior so that it only echoes carriage-return but still passes both carriage-return and line-feed to the program. This function can be used to advantage with programs that do cursor positioning and which do not want line-feed echoed because it might cause the screen display to scroll up a line. The "Z" function restores the line-feed echoing to its normal mode.

7. TSX-Plus EMT'S

TSX-Plus provides several system service calls (EMTs) in addition to those compatible with RT-11. In order to take advantage of the special features of TSX-Plus, programs written to run under both TSX-Plus and RT-11 should check to see if they are under TSX-Plus. This chapter describes the preferred method of checking and goes on to describe several of the special EMTs provided by TSX-Plus. EMTs which relate specifically to features described elsewhere in this manual are included in the appropriate chapters.

7.1 Determining if a job is running under TSX-Plus

In cooperation with Digital Equipment Corporation, a bit has been allocated in the RT-11 sysgen options word at fixed offset 372 into the RMON. The high order bit (bit 15; mask 100000) of this word will be set (1) if the current monitor is TSX-Plus version 5.0 or later. This bit will be clear if the monitor is any version of RT-11. Testing this bit is the preferred method of determining if a job is running under TSX-Plus. However, if a program is expected to also be used under older versions of TSX-Plus, then an alternative method is necessary. For older versions of TSX-Plus, first issue the .SERR request to trap invalid EMT requests and then issue the TSX-Plus EMT to determine the time-sharing line number. If the job is running under RT-11, this EMT will be invalid and the carry bit (indicating an error) will be set on return. If the job is running under TSX-Plus, then the EMT will return without error and the line number will be in R0.

Example:

```
.TITLE TSXENV
.ENABL LC
;
; Demonstrate preferred method of determining whether job is
; running under TSX-Plus or RT-11
;
.MCALL .PRINT,.EXIT,.SERR,.HERR

JSW      = 44                ;Job Status Word address
RMON     = 54                ;Pointer to base of RMON
SYSGEN   = 372               ;Index into RMON for sysgen features

START:   .PRINT #UNDER      ;"Running under"
;
; This is the preferred method, but will not work prior to
; TSX-Plus version 5.0
;
      MOV     RMON,R1        ;Point to base of RMON
      TST     SYSGEN(R1)     ;See if running under TSX-Plus
      BPL     RT11           ;Branch if running under RT-11
;
; This is the old method, but will work correctly with
; all versions of TSX-Plus
;
      .SERR                ;Trap invalid EMT error
```


TSX-Plus EMTs

```

;      MOV      #TSXLN,R0      ;Point to EMT arg block to
;      EMT      375            ;Determine TSX-Plus line number
;      BCS      RT11           ;Branch if running under RT-11
;
      .PRINT    #TSXPPLS      ;"TSX-Plus"
      .EXIT
RT11:   .HERR      ;Reset SERR trap
      .PRINT    #NOTPLS      ;"RT-11"
      .EXIT

TSXLN:  .BYTE     0,110        ;EMT arg block to get line number
      .NLIST     BEX
UNDER:  .ASCII    /Monitor is /<200>
TSXPPLS: .ASCIZ    /TSX-Plus./
NOTPLS: .ASCIZ    /RT-11./

      .END      START

```

7.2 Determining the TSX-Plus line number

The following EMT will return in R0 the number of the line to which the job is attached. Physical lines are numbered consecutively starting at 1 in the same order as specified when TSX-Plus is generated. Detached job lines occur next and virtual lines are numbered last.

The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,110
```

Example:

```

      .TITLE     LNTT
      .ENABL     LC

; What TSX line number is this terminal attached to?
; And what type terminal does TSX-Plus think it is?

      .MCALL     .PRINT,.EXIT,.TTYOUT,.SERR,.HERR
      .GLOBL     PRTDEC      ;Subroutine to print a word in decimal

                                ;Are we under TSX-Plus?
START:  .SERR      ;Stop error aborts
      MOV      #TSXLN,R0      ;Set up EMT request to
      EMT      375            ;Get TSX-Plus line number

```



```

BCS      NOTTSX      ;If error, not under TSX-Plus
MOV      RO,LINE     ;Save it
.HERR     ;Enable error aborts
.PRINT   #LINMSG     ;Display line number message
MOV      LINE,RO     ;Recall line number
CALL     PRTDEC      ;Display line number

.PRINT   #TRMSG      ;Display term type message
MOV      #TTYTYPE,RO ;Set up EMT request to
EMT      375         ;Get terminal type from TSX-Plus
                        ;Returns into RO
ASL      RO          ;Convert to word offset
.PRINT   TYPE(RO)    ;Print type from index into table
.EXIT    ;All done

NOTTSX: .PRINT #TSXERR ;Say we are not under TSX-Plus
.EXIT

LINE:   .WORD    0      ;Storage for TSX line number
TERM:   .WORD    0      ;Storage for TSX term type code
TSXLN:  .BYTE    0,110  ;TSX line number EMT parameters
TTYTYPE: .BYTE    0,137 ;TSX term type EMT parameters

; Table of pointers to TSX term type names
.EVEN
TYPE:   .WORD    UNK,VT52,VT100,HAZEL,ADM3A,LA36,LA120,DIABLO,QUME
.NLIST  BEX
LINMSG: .ASCII   /TSX-Plus line number: /<200>
TRMSG:  .ASCII   <15><12>/Terminal type: /<200>
TSXERR: .ASCIZ   /?LNTT-F-Not running under TSX-Plus/
UNK:    .ASCIZ   /Unknown/
VT52:   .ASCIZ   /VT-52/
VT100:  .ASCIZ   /VT-100/
HAZEL:  .ASCIZ   /Hazeltime/
ADM3A:  .ASCIZ   /ADM3A/
LA36:   .ASCIZ   /LA36/
LA120:  .ASCIZ   /LA120/
DIABLO: .ASCIZ   /Diablo/      ;Diablo and Qume are equivalent
QUME:   .ASCIZ   /Qume/        ;Diablo and Qume are equivalent

.END    START

```

7.3 Determining the terminal type

The following EMT will return in RO a value that indicates what type of time-sharing terminal is being used with the line. The form of the EMT is:

TSX-Plus EMTs

EMT 375

with R0 pointing to the following argument block:

.BYTE 0,137

The terminal type is specified either when the TSX-Plus system is generated or by use of the SET TT command (e.g., SET TT VT100). The terminal type codes which are currently defined are listed below. The types Diablo and Qume are functionally equivalent.

Terminal-type	Code
-----	----
(Unknown)	0
VT52	1
VT100	2
Hazeltine	3
ADM3A	4
LA36	5
LA120	6
Diablo & Qume	7

A type code of 0 (zero) is returned if the terminal type is unknown.

Example:

See the example program LNTT in the section on determining the TSX-Plus line number.

7.4 Determining or changing the user name

When using the LOGON system access program, each user is assigned both a user name and a project, programmer number. TSX-Plus provides an EMT which allows an application program to obtain the user name or (with operator privilege) to change it. User names may be up to twelve characters in length. If the LOGON program is not used, the user name will initially be blank, although it may be changed to a non-blank name. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block to determine the user name:

.BYTE 0,147
.WORD buff-addr

where "buff-addr" is a pointer to a 12 byte area to contain the user name which is returned.

To change the current user name, R0 should instead point to the following argument block:


```
.BYTE 1,147
.WORD buff-addr
```

where "buff-addr" is a pointer to a 12 byte area containing the new user name. Operator privilege is required to change the user name. If changing the user name is attempted without operator privilege, the the name will not be changed and the carry bit will be set on return.

Example:

```
.TITLE GSUNAM
.ENABL LC
```

; Demonstrate TSX-Plus EMT to get/set user name

```
ERRBYT = 52 ;EMT error code location
```

```
.MCALL .PRINT,.EXIT
```

```
START: .PRINT #NAMEIS ;Preface user name
MOV #GSUNAM,R0 ;Point to EMT arg block to
EMT 375 ;Get user name
.PRINT #NAMBUF ;And display it
```

```
MOV #NEWNAM,NAMADD ;Point to new user name
INCB GSUNAM ;Set low bit to set name
MOV #GSUNAM,R0 ;Point to EMT arg block to
EMT 375 ;Set new user name
BCC 1$ ;Error?
```

```
1$: .PRINT #NOPRIV ;Must have operator privilege
.EXIT
```

```
.NLIST BEX
GSUNAM: .BYTE 0,147 ;EMT arg block to get user name
NAMADD: .WORD NAMBUF ;Pointer to receive area
NAMBUF: .BLKW 6 ;Six word name area (12 bytes)
.WORD 0 ;Make it ASCIIZ
NEWNAM: .ASCII /CHAUNCY / ;The new name (12 bytes)
NAMEIS: .ASCII /Your current user name is: /<200>
NOPRIV: .ASCIIZ /Operator privilege necessary to set user name./
.END START
```

7.5 Controlling the size of a job

Under RT-11, the .SETTOP EMT is used to set the top address of a job. The TSX-Plus .SETTOP EMT does not actually alter the memory space allocated to a job but simply checks to see if the requested top of memory is within the region actually allocated to the job and if not returns the address of the top of the allocated job region. The TSX-Plus .SETTOP EMT was implemented this way

TSX-Plus EMTs

because many programs written for RT-11 routinely request all of memory when they start regardless of how much space they actually need.

The memory space actually allocated for a job can be controlled by use of the "MEMORY" keyboard command or by use of the EMT described below. The memory size specified by the most recently executed MEMORY keyboard command is considered to be the "normal" size of the job. The EMT described here can be used to alter the memory space allocated to a job but the job size reverts to the normal size when the job exits or chains to another program.

The form of the EMT used to change a job's size is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,141  
.WORD    top-address
```

where "top-address" is the requested top address for the job. If this address is larger than the allowed size of a job, the job will be expanded to the largest possible size. On return from the EMT, R0 contains the address of the highest available word in the program space.

A program is not allowed to change its size if it was started by use of the "RUN/DEBUG" command or the system was generated without allowing program swapping. In either of these cases the EMT operates exactly like a .SETTOP request (i.e., the requested program top address will not be allowed to exceed the normal program size).

See also the description of the SETSIZ program in Appendix A for information about how the default memory allocation for a program can be built into the SAV file for the program.

Example:

See the example program CKSTAT in the section on determining job status information.

7.6 Obtaining TSX-Plus system values (.GVAL)

The .GVAL EMT that is normally used to obtain RT-11 system values can also be used to obtain TSX-Plus system values. Although a simulated RMON is normally mapped into each job so that it may directly access fixed offsets into RMON, the .GVAL function is the preferred method for obtaining system values. Under TSX-Plus, the simulated RMON need not be mapped into a job's virtual address space (see Chapter 8). The .GVAL EMT will still function correctly even if RMON is not mapped into the job. In addition to the positive offset values which are documented for use with RT-11, the following negative offset values may be used to obtain TSX-Plus system values:

Offset	Value
-2.	Job number
-4.	"Lead-in" character used for terminal control options
-6.	1 if privileged job; 0 if non-privileged job
-8.	1 if PAR 7 mapped to I/O page; 0 otherwise
-10.	Project number job is logged on under
-12.	Programmer number job is logged on under
-14.	TSX-Plus incremental license number
-16.	Current job priority
-18.	Maximum allowed job priority
-20.	Number of blocks per job in SY:TSXUCL.TSX
-22.	Job number of primary line (0 for primary line)
-24.	Name of system device (RAD50) (device on which RT-11 was booted when TSX-Plus was started; may not correspond to current SY assignment)
-26.	Minimum fixed-high-priority value
-28.	Maximum fixed-low-priority value

As with the standard .GVAL function, the system values are returned in R0.

Example:

```
.TITLE TSGVAL
.ENABL LC
```

```
;Demonstrate usage of .GVAL with both positive (RT-11)
; and negative (TSX-Plus) offsets
```

```
.MCALL .GVAL,.PRINT,.EXIT
```

```
.GLOBL PRTDEC      ;Subroutine to print a word in decimal
.GLOBL PRTR50      ;Subroutine to print a RAD50 word
```

```
SYSGEN = 372      ;RMON offset to sysgen options word
```

```
START: .GVAL #AREA,#SYSGEN ;Examine system options word
      TST R0 ;See if we are running TSX-Plus
      BPL 9$ ;Exit if not
      .PRINT #LICENS ;"License # is"
      .GVAL #AREA,#-14. ;Obtain last 4 digits of license #
      CALL PRTDEC ;And display it
      .PRINT #SYSTEM ;"Started from"
      .GVAL #AREA,#-24. ;Get system device
      CALL PRTR50 ;And display it
      .PRINT #JOBNUM ;"Job # is"
      .GVAL #AREA,#-2 ;Get TSX-Plus job number
      CALL PRTDEC ;Display the job number
      .GVAL #AREA,#-22. ;See if this is the primary line
      TST R0 ;0 if primary
```


TSX-Plus EMTs

```
          BEQ      9$              ;Done if so
          .PRINT   #VIRT           ;Else say virtual job
9$:       .EXIT

AREA:     .BLKW    2              ;2 word EMT arg area
          .NLIST   BEX

LICENS:   .ASCII   /TSX-Plus license number /<200>
SYSTEM:   .ASCII   <15><12>/TSX-Plus started from /<200>
JOBNUM:   .ASCII   /:/<15><12>/TSX-Plus line number /<200>
VIRT:     .ASCII   / (This is a virtual line)/<200>

          .END      START
```

7.7 Determining job status information

The information about various jobs on the system which is displayed by the SYSTAT command may also be obtained by application programs. An EMT is provided with several subfunctions to obtain the desired job status information. This EMT may obtain information about any job on the system, not only itself. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 0,144
.BYTE line-#,sub-function
.WORD buf-address
```

where "line-#" is the number of the time-sharing line about which information is to be returned. Line numbers are in the range 1 up to the highest valid line number for the system. "Sub-function" is a function code which indicates the type of information to be returned by the EMT (see below). "buf-address" is the address of the first word of a 2 word buffer area into which the returned value is stored. Note: some of the functions only return a single word value in which case the value is returned into the first word of the buffer area.

If an error occurs during the execution of the EMT, the carry-flag is set on return and the following error codes indicate the type of error:

Errors:

<u>Code</u>	<u>Meaning</u>
0	Indicated line number is not currently logged on.
1	Invalid sub-function code.
2	Invalid line number (0 or higher than largest valid line number).

Each of the sub-functions is described below:

Subfunction # 0 -- Check status of line. The value returned contains bit flags that indicate the status of the job. The following bit flags are defined:

000001 = This is a virtual line.
 000002 = This is a detached job line.
 000100 = Job has locked itself in memory.
 000200 = Job has operator privilege.

Subfunction # 1 -- Get job's execution state. This subfunction returns a code that indicates a job's current execution state. The following code values are defined:

1 = Non-interactive high priority run state.
 2 = Normal priority run state.
 3 = Fixed-low-priority run state.
 4 = Waiting on input from the terminal.
 5 = Waiting for output to be written to terminal.
 6 = Doing a timed wait.
 7 = Suspended because .SPND EMT done.
 8 = Waiting for access to a shared file.
 9 = Waiting for a inter-job message.
 10 = Waiting for access to USR (file management) module.
 11 = Waiting for non-terminal I/O to finish.
 12 = Waiting for access to spool file.
 13 = Interactive high priority run state.
 14 = Fixed-high-priority run state.
 15 = Waiting for memory expansion.

Subfunction # 2 -- Determine amount of memory used by job. This function returns the number of 256-word blocks of memory that are currently being used by the job, including PLAS regions.

Subfunction # 3 -- Determine connect time for job. This function returns the number of minutes that a job has been logged onto the system.

Subfunction # 4 -- Determine position of job in memory. This function returns the 256-word block number of the start of the memory area allocated to the job.

Subfunction # 5 -- Get name of program being run by job. This function returns a 2 word value. The two words contain the RAD50 value for the name of the program currently being run by the job.

Subfunction # 6 -- Get project and programmer number for job. This function returns a two word value. The first word contains the project number that the job is logged on under; the second word contains the programmer number.

TSX-Plus EMTs

Subfunction # 7 -- Get CPU time used by job. This function returns a two word value that contains the number of clock-ticks of CPU time used by the job. The first word contains the high-order 16-bits of the value, the second word contains the low-order 16-bits.

Subfunction # 8 -- Get current job execution priority. This function returns one word that contains the current job execution priority level (0-127).

Example:

```
.TITLE  CKSTAT
.ENABL  LC
```

; Demonstration of MEMTOP, JSTAT and SNDMSG EMTs of TSX-Plus

```
ERRBYT = 52                ;EMT error code location
PRGNAM  = 5                ;JSTAT subfunction code to get prog. name
```

```
.MCALL  .TWAIT,.EXIT
```

```
START:  MOV    #MEMTOP,RO    ;Point to EMT arg block to
      EMT      375          ;Set job size
;Only works in swapping environment, otherwise behaves like .SETTOP
      CMP      RO,#HILIM    ;See if we got what we wanted
      BHIS     AGAIN        ;Go on if so
      .EXIT          ;else quit (can't disp err msg from det line)
```

```
AGAIN:  MOVVB   #1,LINE      ;Check all lines starting with #1
CHECK:  MOV      #JSTAT,RO    ;Point to EMT arg block
      EMT      375          ;Get name of job being run
      BCS      ERRTP        ;Go find out what kind of error
      CMP      BUFADD,DUNJUN ;Is this line goofing off?
      BNE      NEXT        ;No, proceed
      CMP      BUFADD+2,DUNJUN+2 ;May be, check for sure
      BNE      NEXT        ;No, proceed
;Send a message to the offending line
;(Each message must be < 88. bytes)
      MOVVB    LINE,YOOHOO    ;Who is the guilty party?
      MOV      #MESAG1,MSGADD ;Prepare part one of message
      MOV      #SEND,RO       ;Point to EMT arg block to
      EMT      375          ;Send a message to that line
      MOV      #MESAG2,MSGADD ;Prepare for part two of the message
      MOV      #SEND,RO       ;Point to EMT arg block to
      EMT      375          ;Send part 2 of message
      MOV      #MESAG3,MSGADD ;Prepare for part three of the message
      MOV      #SEND,RO       ;Point to EMT arg block to
      EMT      375          ;Send part 3 of message
      MOV      #MESAG4,MSGADD ;Prepare for part four of the message
      MOV      #SEND,RO       ;Point to EMT arg block to
      EMT      375          ;Send part 4 of message
```



```

NEXT:  INCB    LINE      ;Try next line
      CMPB    LINE,MAXLIN ;Have we checked them all?
      BGT     SLEEP      ;Yes, wait awhile
      BR      CHECK      ;Go check the rest of the lines
SLEEP:  .TWAIT  #AREA,#TIME ;Come back in 5 minutes
      BR      AGAIN      ;And try again
ERRTYP: CMPB    @#ERRBYT,#1 ;Which error is it
      BLT     NEXT      ;0 --> line not logged on, try next line
      BEQ     2$         ;1 --> invalid sub-function code, give up
      MOVB    LINE,MAXLIN ;2 --> line > last valid line
      DECB    MAXLIN     ;Largest valid line number
      BR      SLEEP      ;should only happen first time

2$:     .EXIT           ;Invalid code should never happen
                        ;Might as well kill job
MEMTOP: .BYTE    0,141   ;Argument block for MEMTOP EMT
      .WORD    HILIM     ;Upper address limit
JSTAT:  .BYTE    0,144   ;Argument for JSTAT EMT
LINE:   .BYTE    0       ;TSX-Plus line number to be checked
SUBFUN: .BYTE    PRGNAM   ;EMT subfunction
      .WORD    BUFADD     ;Address of 2-word buffer for returned value
BUFADD: .BLKW    2        ;2 word buffer to hold stat result
MAXLIN: .BYTE    30.      ;Maximum number of lines under TSX-Plus
      .EVEN      ;Will be altered to max valid line #
SEND:   .BYTE    0,127   ;EMT arg block to send a message
YOOHOO: .WORD    0        ;Destination line number
MSGADD: .WORD    MESAG1   ;Message to be sent
AREA:   .BLKW    2        ;.TWAIT arg area
TIME:   .WORD    0        ;time high word
      .WORD    5*60.*60.  ;5min * 60.sec/min * 60.ticks/sec
      .NLIST    BEX
DUNJUN: .RAD50    /DUNJUN/ ;Name of illicit program
MESAG1: .ASCII    <7><15><12>
      .ASCII    /*****<15><12>
      .ASCIZ    /*<15><12>
MESAG2: .ASCII    /* Continued use of this system */<15><12>
      .ASCIZ    /* for game playing will result */<15><12>
MESAG3: .ASCII    /* in loss of user privileges!! */<15><12>
      .ASCIZ    /*<15><12>
MESAG4: .ASCIZ    /*****<15><12><7>

HILIM:  .END      START

```

7.8 Job monitoring

A monitoring watch may be established to allow a job to monitor the status of other time-sharing jobs. For example, if a detached job were being used as a common file server for several other jobs, it would be useful to know if a served program with a pending request aborts. The outstanding request could then be purged. The monitoring facility may also be used in lieu of message

TSX-Plus EMTs

channels (see Chapter 10) when only a small amount of information needs to be communicated (e.g. one word).

When a job is being monitored, the operating system will report certain job status changes, such as logging on, starting or exiting a program, or logging off. In addition, the monitored job itself may issue status reports to any job which may be monitoring it. After a job has established a monitor watch for a given line, a completion routine is entered in the monitoring job whenever a monitor status report is issued for the monitored line, whether the report was issued by the monitored job itself or by the system because of a status change. Job monitoring may be used effectively in conjunction with inter-job message communications and with detached jobs.

There are three system service calls related to job monitoring. All have the form:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE    n,157
.WORD    job-number
.WORD    completion-routine
```

where "n" designates which job monitoring function is to be performed, "job-number" designates the line number which is to be monitored, and "completion-routine" is the address of a completion routine in the monitoring job which is to be entered whenever a monitor status report is generated for the line being monitored ("job-number"). The job-number may designate any primary time-sharing line, detached job or virtual job number. It may not refer to a dedicated CL line. Job-numbers (lines) are assigned in the following order: time-sharing lines in the order they were declared during system generation, starting with number 1; detached job lines; virtual job lines. Detached and virtual jobs are assigned to line numbers in the order they were activated. Unused detached lines are reserved and are never assigned to virtual jobs.

7.8.1 Establishing a monitoring connection:

The form of the argument block to establish a monitoring connection with a line is:

```
.BYTE    0,157
.WORD    job-number
.WORD    completion-routine
```


Errors:

<u>Code</u>	<u>Meaning</u>
1	Invalid job number specified
2	No free job monitoring control blocks (increase TSGEN parameter MAXMON)

The specified completion routine will be entered whenever a monitor status report is issued for the specified job-number. On entry to the completion routine, the low order byte of R0 contains the line number of the job originating the monitor status report. Note that the same completion routine may be specified when monitoring more than one job. The high-order bit (mask 100000) of R0 will be clear (0) if the monitor status was originated by the system and will be set if the monitored job itself issued the status. R1 will contain a 16-bit status value. Status values generated by the system for monitored lines are:

<u>Status Code</u>	<u>Meaning</u>
1	Job has been initialized
2	Job has logged on using the LOGON program
3	Job has started running a program
4	Job has returned control to the keyboard monitor
5	Job has logged off

Status code 1 is generated when a logged off line is started, usually by typing a carriage-return at that terminal. Status code 3 is generated whenever a program is either run or chained to. Status code 4 is generated whenever a program exits or is aborted, but not when it chains to another program.

7.8.2 Cancel a monitoring connection:

The form of the argument block to cancel a monitoring connection with a line is:

```
.BYTE 1,157
.WORD job-number
```

If "job-number" is zero (0), then all job monitoring connections established by the monitoring job (the job issuing this EMT) are cancelled. All job monitoring requests are also cancelled if the job exits, aborts, chains or issues a .SRESET or .HRESET.

Errors:

<u>Code</u>	<u>Meaning</u>
1	Invalid job number specified

7.8.3 Broadcast status report to monitoring jobs:

The form of the argument block to broadcast a monitor status report is:

```
.BYTE 2,157
.WORD status-value
```

where "status-value" is a 16-bit value to be broadcast to all jobs which are monitoring the job which broadcasts the status-value (the job issuing this EMT). On entry to the completion routine in the monitoring job(s), the status-value is in R1, the low byte of R0 contains the job-number of the job broadcasting the status-value, and the high-order bit of R0 (mask 100000) is set (1).

Errors:

Code	Meaning
----	-----
0	No jobs are monitoring this line

Example:

```
.TITLE MONCPL
.ENABL LC
;
; Demonstrate job monitoring completion routines.
; Watch a dial-in line and announce when it is activated.
;
.MCALL .PRINT,.EXIT,.SPND,.RSUM
.DSABL GBL
.GLOBL R5OASC

ERRBYT = 52 ;EMT error byte address
MONLIN = 8. ;Line number to be monitored
NFYLIN = 1. ;Line number to be notified

START: MOV #MONJOB,R0 ;Point to EMT arg block to
      EMT 375 ;Schedule job monitor compl rtn
      BCC 1$ ;Branch if OK

; Job monitoring scheduling error
      MOVB @#ERRBYT,R1 ;Get EMT error code
      ADD #^O,R1 ;Convert to ASCII
      MOVB R1,ERRCOD ;Stuff into error message
4$: .PRINT #CPLERR ;Report error
      .EXIT ;And abort

;
; Wait for event on monitored line
;
1$: .SPND ;Wait for event
      BR 4$ ;Only get here on error
```



```
;
; Completion routine which is entered when event occurs on monitored line
;
```

```
MONCPL: MOV     RO,SENDER      ;Remember whose monitor report
        MOV     #MONJOB,RO    ;Point to EMT arg block to
        EMT     375           ;Reschedule myself
        BCC     1$            ;Branch if OK
        MOVB    @#ERRBYT,RO   ;Get error code
        ADD     #^0,RO        ;Convert to ASCII
        MOVB    RO,ERRCOD     ;Save error code
        .RSUM               ;Restart mainline and report error
        BR      3$            ;Abort completion routine
1$:     TST     SENDER        ;Was this a system generated message?
        BMI     3$            ;Ignore if not
        CMP     R1,#3         ;Running program?
        BNE     2$            ;Branch if not
        MOV     #GPRGNM,RO    ;Point to EMT arg block to
        EMT     375           ;Get program name
; Note that a short program may already have returned to KMON by now!
        MOV     #R50BLK,RO    ;Point to program name
        CALL    R50ASC        ;Convert into ASCII in message
2$:     DEC     R1            ;0 index status code
        MUL     #CODLEN,R1    ;Convert status code to message offset
        ADD     #CODBEG,R1    ;Convert to message address
        MOV     R1,MSADR      ;Point to correct status message
        MOV     #NOTIFY,RO    ;Point to EMT arg block to
        EMT     375           ;Send notification to operator
3$:     RETURN
```

```
;
; EMT argument blocks and word buffers
;
```

```
MONJOB: .BYTE   0,157         ;EMT arg block to monitor a line
        .WORD   MONLIN        ;Line (job) number to be monitored
        .WORD   MONCPL        ;Address of completion routine
NOTIFY: .BYTE   0,127         ;EMT arg block to send text
        .WORD   NFYLIN        ;Line number to be notified
MSADR:  .WORD   0              ;Address of message to be sent
GPRGNM: .BYTE   0,144         ;EMT arg block to get program name
        .BYTE   MONLIN,5      ;Line number, sub-function
        .WORD   R50NAM        ;Address to put RAD50 name
R50BLK: .WORD   R50NAM        ;Address of input buffer (RAD50)
        .WORD   PRGNAM        ;Address of output buffer (ASCII)
        .WORD   6              ;Number of chars to convert
R50NAM: .WORD   0,0           ;RAD50 value of program name
SENDER: .WORD   0              ;Copy of sending infor
```

```
;
; Text and byte buffers
;
        .NLIST  BEX
```


TSX-Plus EMTs

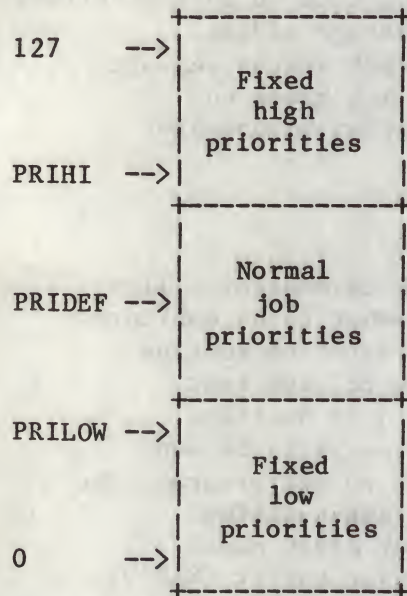
```

AC      = .
COBEG: .ASCIZ <15><12><7>/Job has been initialized      /
CODLEN  = . - AC
        .ASCIZ <15><12><7>/Job has completed LOGON      /
        .ASCII <15><12><7>/Job executing program /
PRGNAM: .ASCIZ                                          /PRGNAM /
        .ASCIZ <15><12><7>/Job returned to KMON          /
        .ASCIZ <15><12><7>/Job logged off                /
CPLERR: .ASCII /Job monitoring completion routine error /
ERRCOD: .ASCIZ /0/
        .END      START

```

7.9 Setting job priority

Jobs may be assigned priority values in the range 0 to 127 to control their execution scheduling relative to other jobs. The priority values are arranged in three groups: the fixed-low-priority group consists of priority values from 0 up to the value specified by the PRILOW sysgen parameter; the fixed-high-priority group ranges from the value specified for the PRIHI sysgen parameter up to 127; the middle priority group ranges from (PRILOW+1) to (PRIHI-1). The following diagram illustrates the priority groups:



Job scheduling is performed differently for jobs in the fixed-high-priority and fixed-low-priority groups than for jobs with normal interactive priorities. Jobs with priorities in the fixed-low-priority group (0 to PRILOW) and the fixed-high-priority group (PRIHI to 127) execute at fixed priority values. That is, the priority absolutely controls the scheduling of the job for execution relative to other jobs. A job with a fixed priority is allowed to execute as long as it wishes until a higher priority job becomes active.

The fixed-high-priority group is intended for use by real-time programs. The fixed-low-priority group is intended for use by very low priority background tasks. Normal time-sharing jobs should not be assigned priorities in either of the fixed priority groups.

The middle group of priorities from (PRILOW+1) to (PRIHI-1) are intended to be used by normal, interactive, time-sharing jobs. Jobs with these assigned priorities are scheduled in a more sophisticated manner than the fixed-priority jobs. In addition to the assigned priority, external events such as terminal input completion, I/O completion, and timer quantum expiration play a role in determining the effective scheduling priority.

When a job with a normal priority switches to a virtual line, the priority of the disconnected job is reduced by the amount specified by the PRIVIR sysgen parameter. This causes jobs that are not connected to terminals to execute at a lower priority than jobs that are. This priority reduction does not apply to jobs with priorities in the fixed-high-priority group or the fixed-low-priority group. The priority reduction is also constrained so that the priority of jobs in the normal job priority range will never be reduced below the value of (PRILOW+1).

The following EMT can be used to set the job priority from within a program. The job priority can also be set from the keyboard with the SET PRIORITY command. The current job priority, maximum allowed priority, and fixed-high- and fixed-low-priority boundaries may be determined with the .GVAL request. See the TSX-Plus System Manager's Guide for more information on the significance of priority in job scheduling. The form of this EMT is:

EMT 375

with R0 pointing to the following EMT argument block:

```
.BYTE  0,150
.WORD  value
```

where "value" is the priority value for the job. The valid range of priorities is 0 to 127 (decimal). The maximum job priority may be restricted by the system manager. If a job attempts to set its priority above its maximum allowed priority, its priority will be set to the maximum allowed. This EMT does not return any errors.

Example:

```
.TITLE  GSPRI
.ENABL  LC
```

; Demonstrate EMT to set job priority

```
.MCALL  .GVAL,.GTIN,.PRINT,.EXIT
.GLOBL  PRTDEC
```


TSX-Plus EMTs

```

CURPRI = -16.           ;GVAL offset to get current priority
MAXPRI = -18.           ;GVAL offset to get maximum priority

START: .PRINT #CURIS      ;"current priority is"
       .GVAL #AREA,#CURPRI ;Obtain current job priority in R0
       MOV   R0,R1        ;Save it
       CALL  PRTDEC        ; and display it
       .PRINT #MAXIS      ;"maximum priority is"
       .GVAL #AREA,#MAXPRI ;Obtain maximum allowable job priority
       MOV   R0,R2        ;Save it
       CALL  PRTDEC        ; and display it
       ADD   #10.,R1       ;Try to boost priority by 10
       CMP   R1,R2        ;Unless exceeds maximum
       BLE   1$           ;Use 10 larger if <= maxpri
       MOV   R2,R1        ;Else use maxpri
1$:    MOV   R1,NEWPRI      ;Set new priority in EMT arg block
       MOV   #SETPRI,R0    ;Point to EMT arg block to
       EMT   375           ;Sset new job priority
       .PRINT #NEWIS      ;"new priority is"
       .GVAL #AREA,#CURPRI ;Obtain new priority
       CALL  PRTDEC        ; and display it
       .EXIT

AREA:  .BLKW  10           ;General EMT arg block
SETPRI: .BYTE 0,150        ;EMT arg block to set job priority
NEWPRI: .WORD 50.         ;New job priority goes here

       .NLIST BEX
CURIS: .ASCII /Current job priority = /<200>
MAXIS: .ASCII <15><12>/Maximum job priority = /<200>
NEWIS: .ASCII <15><12>/- New - job priority = /<200>

       .END   START

```

7.10 Forcing [non]interactive job characteristics

The following EMT can be used to cause a job to be scheduled either as an interactive job or as a non-interactive job. Programs which do a large amount of terminal input, but which are not truly interactive jobs in the usual sense, such as file transfer programs, should use this EMT to avoid excessive interference with normal interactive time-sharing jobs. This feature may also be selected with the R[UN]/NONINTERACTIVE command. See the TSX-Plus System Manager's Guide for more information on job scheduling and the significance of interactive vs. non-interactive jobs.

The form of this EMT is:

```
EMT      375
```


with R0 pointing to the following argument block:

```
.BYTE 0,153
.WORD mode
.WORD 0
```

If the value of "mode" is 0, then the job will never be scheduled as an interactive job. If "mode" is 1, then the job will be scheduled as other interactive jobs are, dependent on terminal input.

Example:

```
.TITLE NONINT
.ENABL LC
```

; Demonstrate EMT to schedule job as interactive or non-interactive

```
.MCALL .TTYIN,.TTYOUT,.PRINT,.EXIT
```

```
JSW      = 44                ;Job Status Word address
TTSPC    = 10000             ;TT special mode bit (single-char)
CTRLZ    = 32                ;ASCII CTRL-Z (move on command)

START:   MOV      #SINGLE,R0   ;Point to EMT arg block to
        EMT      375          ;Turn on single character activation
        BIS      #TTSPC,@#JSW ;Finish turning on single char mode

        MOV      #NONINT,R0   ;Point to EMT arg block to
        EMT      375          ;Schedule this as non-interactive job

1$:      .PRINT   #SLOW        ;"May be slow now if system busy"
        .TTYIN    ;Get a char
        CMPB     R0,#CTRLZ    ;If CTRL-Z
        BEQ      2$          ;Then move on
        .TTYOUT   ;Else echo it back (we have to echo
        ; when in single char mode)
        BR       1$          ;And repeat

2$:      MOV      #1,SELECT    ;Want to be interactive now
        MOV      #NONINT,R0   ;Point to EMT arg block to
        EMT      375          ;Schedule this as an interactive job
        .PRINT   #FAST        ;"See how much faster now"
3$:      .TTYIN    ;Get a char
        CMPB     R0,#CTRLZ    ;If CTRL-Z
        BEQ      4$          ;Then move on
        .TTYOUT   ;Else echo it back
        BR       3$          ;And repeat

4$:      .EXIT
```


TSX-Plus EMTs

```
SINGLE: .BYTE 0,152      ;EMT arg block to set term option
        .WORD 'S        ;Single char activation
        .WORD 0
NONINT: .BYTE 0,153      ;EMT arg block to sched as [non]interactive
SELECT: .WORD 0          ;Initially make non-interactive
        .WORD 0
        .NLIST BEX
SLOW:   .ASCII /Type some characters in now. If the system has several /
        .ASCII /interactive jobs/<15><12>
        .ASCII /response will be slow. (Control-Z to get out /
        .ASCIZ /of this mode.)/
FAST:   .ASCIZ <15><12>/Try again. Response should be much better./
        .END START
```

7.11 Sending a message to another line

The following EMT can be used to cause a message to display on another line's terminal. (This is a different feature than message communication channels.) The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE 0,127
.WORD line-number
.WORD message-address
```

where "line-number" is the number of the line to which the message is to be sent and "message-address" is the address of the start of the message text that must be in ASCIZ form. The message length must be less than 88 bytes.

Example:

See the example program CKSTAT in the section on determining job status information.

7.12 Mount a file structure

This EMT is used to tell TSX-Plus that a file structure is being mounted and that TSX-Plus should begin caching the file directory for the device. The effect of this EMT is the same as doing a system MOUNT keyboard command. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 0,134
.WORD device-spec-address
.WORD 0
```

where "device-spec-address" is the address of a word containing the RAD50 form of the name of the device on which the file structure is being mounted. If there is no room left in the table of mounted devices, the carry bit is set on return and the error code returned is 1.

Example:

```
.TITLE MOUNT
.ENABL LC
```

; Demonstrate TSX-Plus EMT to "MOUNT" (do directory caching on) a device

```
BS      .GLOBL  IRAD50          ;SYSLIB RAD50 conversion subroutine
        = 10          ;ASCII Backspace
        .MCALL  .PRINT,.GTLIN,.EXIT

START:  .GTLIN  #BUFFER,#PROMPT ;Ask for name of device
        MOV     #R50BLK,R5      ;Point to arg block for next call
        CALL    IRAD50          ;Convert ASCII device name to RAD50
        MOV     #MOUNT,R0       ;Point to EMT arg block to
        EMT     375             ;Mount a file structure (directory caching)
        BCC     START           ;Ask for more if OK

        .PRINT  #NOGOOD         ;Say it was not good
        .EXIT

        .NLIST  BEX

MOUNT:  .BYTE    0,134          ;EMT arg block to mount a file structure
        .WORD    DEVNAM        ;Pointer to RAD50 name of device
        .WORD    0             ;Required 0 argument
R50BLK: .WORD    3              ;Number of args for IRAD50 call
        .WORD    THREE         ;Pointer to number of chars to convert
        .WORD    BUFFER        ;Pointer to chars to convert
        .WORD    DEVNAM        ;Pointer to RAD50 name of device
THREE:  .WORD    3              ;Number of chars to convert
DEVNAM:  .WORD    0             ;RAD50 representation of device name
BUFFER:  .BLKB   80.           ;GTIN input buffer
PROMPT:  .ASCII  /Name of device to be mounted:  :/<BS><BS><BS><BS><200>
NOGOOD:  .ASCIIZ /Attempt to MOUNT too many devices./<7>
        .END    START
```


TSX-Plus EMTs

7.13 Dismount a file structure

This EMT can be used to tell TSX-Plus to stop doing directory caching on a particular drive. The effect of this EMT is the same as a DISMOUNT keyboard command. The form of the EMT is:

EMT 375

with R0 pointing to an argument block of the following form:

```
.BYTE 0,135
.WORD device-spec-address
.WORD 0
```

where "device-spec-address" is the address of a word containing the RAD50 name of the device to be dismounted.

Example:

```
.TITLE DISMNT
.ENABL LC
```

; Demonstrate TSX-Plus EMT to "DISMOUNT" (stop caching on) a device

```
BS      .GLOBL  IRAD50          ;SYSLIB RAD50 conversion subroutine
        = 10          ;ASCII Backspace
        .MCALL  .GTIN

START:  .GTIN  #BUFFER,#PROMPT ;Ask for name of device
        MOV    #R50BLK,R5      ;Point to arg block for next call
        CALL   IRAD50          ;Convert ASCII device name to RAD50
        MOV    #DISMNT,R0      ;Point to EMT arg block to
        EMT     375            ;dismount a file structure (stop caching)
        BR     START           ;Repeat (no errors returned)

        .NLIST  BEX

DISMNT: .BYTE 0,135            ;EMT arg block to dismount a file structure
        .WORD  DEVNAM          ;Pointer to RAD50 name of device
        .WORD  0               ;Required 0 argument
R50BLK: .WORD  3               ;Number of args for IRAD50 call
        .WORD  THREE           ;Pointer to number of chars to convert
        .WORD  BUFFER          ;Pointer to chars to convert
        .WORD  DEVNAM          ;Pointer to RAD50 name of device
THREE:  .WORD  3               ;Number of chars to convert
DEVNAM: .WORD  0               ;RAD50 representation of device name
BUFFER: .BLKB  80.             ;GTIN input buffer
PROMPT: .ASCII /Name of device to be dismounted: :/<BS><BS><BS><BS><200>

        .END  START
```


7.14 Set terminal read time-out value

This EMT can be used to specify a time-out value that is to be applied to the next terminal input operation. This EMT allows you to specify the maximum time that will be allowed to pass between the time that you issue a command to get input from the terminal and the time that an activation character is received to terminate the input field. You also specify with this EMT a special activation character that is returned as the terminating character for the field if the input operation times out without receiving an activation character from the terminal. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,117
.WORD     time-value
.WORD     activation-character
```

where "time-value" is the time-out value specified in 0.5 second units and "activation-character" is a single character value that is to be returned as the last character of the field if a time-out occurs. The time value specified with this EMT only applies to the next terminal input field. The time value is reset when the next field is received from the terminal or the time-out occurs. A new time-out value must be specified for each input field that is to be time controlled.

Example:

```
.TITLE   DUNJUN

; Demonstrate use of terminal input time-out testing

.MCALL   .TTYOUT,.TTYIN,.EXIT,.PRINT

START:   .TTYOUT #'?

1$:      MOV      #SETTTO,R0      ;Point to EMT arg block to
      EMT        375              ;Set terminal input time-out
      .TTYIN      ;Get a character from the terminal
      CMP        R0,#<15>         ;Skip over carriage returns
      BEQ        1$
      CMP        R0,#<12>         ;and line feeds
      BEQ        START           ;prompt for next char
      CMP        R0,#^Q          ;Should we quit?
      BNE        1$              ;No, get next char
      .PRINT     #DONE           ;Quit or time-out

      .EXIT                    ;Bye
```


TSX-Plus EMTs

```
SETTTO: .BYTE    0,117          ;EMT arg block
        .WORD    6*60.*2       ;6.min * 60.sec/min * 2.half-sec-units/sec
        .WORD    'Q           ;Activation character on time-out

        .NLIST   BEX
DONE:   .ASCIZ   /STOP - /
        .END     START
```

See also the example program CKTTIE in the section on checking for terminal input errors.

7.15 Establishing break sentinel control

The following EMT can be used to declare a completion routine that will be triggered when the "Break" key is pressed. The form of the EMT is

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,133
.WORD    brkchr
.WORD    cplrtn
```

where "brkchr" is a user defined character that is to be declared the "Break" character and "cplrtn" is the address of the completion routine that is to be called when the break character is received from the terminal. The specified completion routine will be called if the user presses either the key labeled "BREAK" (which transmits a long space) or types the character that is declared as the user-specified break character (brkchr). If no user-specified break character is wanted, specify the value 0 (zero) for "brkchr" in the argument block and only the real "BREAK" key will be activated. Note that on some systems the console terminal "BREAK" key causes entry to the hardware ODT module and for this reason cannot be used with this TSX-Plus function. Only one break routine may be specified at a time for each user. If a break routine was previously specified, it is cancelled when a new routine is declared. If an address of 0 (zero) is specified as the address of the completion routine (cplrtn), any previously specified break routine is cancelled and the break key connection is cancelled. A break routine can be used to signal an asynchronous request for service to a running program. A good example of its use would be to trigger entry to an interactive debugging program.

Example:

```
.TITLE   BRKSNT

;Demo use of break sentinel control

.MCALL   .PRINT,.TWAIT,.EXIT
```



```

.ENABL LC

START: MOV    #BRKSNT,RO    ;Point to argument area to
      EMT     375          ;Establish break sentinel control

      .PRINT  #MESSAG      ;Prompt for key
      .WAIT   #AREA,#TIME  ;Give the user 2 seconds to hit the break key

      TST     YES          ;Ever see a break?
      BNE     DONE         ;Yes, all done
      .PRINT  #NOBRK       ;No, never saw it

DONE:  .EXIT

CMPRTN: .PRINT  #GOTBRK     ;Say we caught the break
      MOV     #1,YES       ;Remember it
      RETURN                    ;And continue
      ;Completion routines are ALWAYS exited with
      ;RTS PC under TSX-Plus, NEVER via RTI

BRKSNT: .BYTE   0,133      ;EMT arg value block to break sentinel control
      .WORD   0            ;Declare only 'BREAK' key as break char
      .WORD   CMPRTN       ;Address of completion routine to be called
      ;when system notices break

YES:    .WORD   0          ;Flag for break seen

AREA:    .BLKW  2          ;2 word arg area for .WAIT

TIME:    .WORD   0          ;high word of time
      .WORD   2.*60.       ;2 sec * 60.tics/sec

      .NLIST  BEX
MESSAG:  .ASCIZ  /You have 2 seconds to hit the break key./
GOTBRK:  .ASCIZ  <15><12>/Break key pressed./
NOBRK:   .ASCIZ  /Never saw the break key./

      .END    START

```

7.16 Checking for terminal input errors

The following EMT can be used to determine if any terminal input errors have occurred. The form of the EMT is:

TSX-Plus EMTs

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 0,116
```

On return from the EMT, the carry-flag is set if an input error has occurred since the line logged on or since the last time a check was made for input errors. The two types of errors that are monitored by this EMT are hardware reported errors (parity, silo-overflow, etc.) and characters lost due to TSX-Plus input buffer overflow.

Example:

```
.TITLE CKTTIE
.ENABL LC

;Check for terminal input errors

.MCALL .PRINT,.TTYIN,.EXIT

START: .PRINT #PROMPT          ;Ask to overflow buffer
        MOV     #100.,R1        ;Set up counter for input loop
        MOV     #SETTTO,R0      ;Point to EMT arg block to
        EMT     375             ;Set terminal time out for 0.5 secs
;Note that this is reset after every activation character!!!
;Start requesting characters. Input characters are stacked in the user
;input buffer until an activation character is seen (e.g. carriage return).
;So, all we have to do to overflow is enter more than the input buffer
;size (defined in TSGEN either by DINSPC or with the BUFSIZ macro)
;and type in too many before activating.
;Use a time-out so we don't have to hit return.
l$:     .TTYIN                  ;Get a character from the terminal
        CMPB    R0,#37          ;Was it time-out activation char?
        BEQ     TIMOUT          ;Yes, exit loop
        SOB     R1,l$           ;Repeat for 100. characters
;For a system with input buffer size=100. in TSGEN, we should be
;able to overflow the buffer before we see an activation char
TIMOUT: MOV     #CKTTIE,R0      ;Point to EMT arg block to
        EMT     375             ;Check for terminal input errors
        BCS     HADERR          ;Say we had errors
        .PRINT  #NOERR         ;Say we had no errors
        .EXIT

HADERR: .PRINT  #YESERR         ;Error message
        CMP     R1,#3           ;Did we fill the buffer?
;Note that last two chars of input buffer are reserved
;for activation chars. Any excess input is discarded.
        BLE     TOOMNY         ;Yes, buffer overflow
        .PRINT  #HDWERR        ;No, hardware error message
```



```

.EXIT
TOOMNY: .PRINT  #OVFERR           ;Buffer overflow message
.EXIT

SETTTO: .BYTE   0,117             ;EMT arg block to set terminal time out
        .WORD   20.              ;to 10 seconds (20 half sec units)
        .WORD   37              ;Passed as activation char on time-out
CKTTIE: .BYTE   0,116             ;EMT arg block to check for input errors
        .NLIST  BEX
YESERR: .ASCIZ  <15><12>/There were errors during terminal input./<7>
OVFERR: .ASCIZ  /(Probably input buffer overflow.)/
HDWERR: .ASCIZ  /(Probably hardware error ... parity, stop bits, data bits)/
NOERR:  .ASCIZ  <15><12>/There were no terminal input errors./
PROMPT: .ASCIZ  /Please enter more than 100 input characters and wait. . ./

.END    START

```

7.17 Checking for activation characters

The following EMT can be used to determine if any activation characters have been received by the line but not yet accepted by the program. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,123
```

If there are pending activation characters, the carry-flag is cleared on return from the EMT; if there are no pending activation characters, the carry-flag is set on return from the EMT.

Example:

```

.TITLE  CKACT
.ENABL  LC

;Demonstrate use of check for activation characters

LEADIN  =      35                ;TSX-Plus program controlled terminal
                                ;option lead-in character
.MCALL  .PRINT,.EXIT,.GTLIN,.TWAIT,.TTYOUT

START:  .PRINT  #PROMPT          ;Request some characters
                                ;And disallow deferred echoing
;Do some processing. Simulated here by .TWAIT
        MOV     #80.,R1          ;Line length counter
1$:     .TTYOUT  #^               ;Tick, tock
        DEC     R1               ;End of line?

```


TSX-Plus EMTs

```

        BNE      2$                ;No, go on
        .TTYOUT  #<15>              ;New line
        .TTYOUT  #<12>
2$:     MOV      #80.,R1            ;Reset line length counter
; . . . .                          ;Wait 1 second here
        .WAIT    #AREA,#TIME      Processing . . .
        MOV      #CKACT,R0         ;Point to EMT arg block to
        EMT      375              ;Check for pending activation characters
        BCS      1$                ;Continue if input not complete

        .GTLIN   #BUFFER           ;Collect the pending input
; . . . .                          Do something with it
        CMP      BUFFER,EX         ;Exit command?
        BNE      1$                ;No, continue processing
        .PRINT   #BYE
        .EXIT

AREA:    .BLKW   10.                ;EMT arg block
TIME:    .WORD   0,1.*60.           ;1.sec * 60.tics/sec
CKACT:    .BYTE  0,123              ;EMT arg block for activation char check
BUFFER:   .BLKB  81.                ;Local input buffer

        .NLIST   BEX
        .EVEN
EX:       .ASCII  /EX/
PROMPT:   .ASCII  <LEADIN>/L/      ;Disallow deferred echoing
        .ASCIZ   /Please enter up to 80 characters, then RETURN:/
BYE:      .ASCIZ   /Thank you./

        .END     START

```

7.18 Sending a block of characters to the terminal

The following EMT can be used to efficiently send a block of characters to the terminal. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

        .BYTE    0,114
        .WORD     buffer
        .WORD     count

```

where "buffer" is the address of the buffer containing the characters to be sent and "count" is a count of the number of characters to be sent. This EMT is much more efficient to use than a series of .TTYOUT EMT's -- it has the same efficiency as a .PRINT EMT but it uses a count of the number of characters to send rather than having the character string in ASCII form.

Example:

```
.TITLE  TTOBLK
.ENABL  LC
```

;Demonstration of the use of the TSX-Plus EMT to send a block of
;characters to the terminal.

```
.MCALL  .EXIT
```

```
START:  MOV    #TTOBLK,R0      ;Point to EMT arg block to
      EMT      375            ;Send a block of chars to the terminal
      .EXIT
```

```
TTOBLK: .BYTE    0,114        ;EMT arg block to send a block of chars
      .WORD      BUFFER        ;Pointer to character buffer
      .WORD      <BUFEND-BUFFER> ;Count of characters to be output
      .NLIST     BEX
```

```
BUFFER: .ASCII   /This EMT is used to send a block of characters /
      .ASCII   /to the terminal./<15><12>
      .ASCII   /It is similar to .PRINT, except that it uses /
      .ASCII   /a count of characters/<15><12>
      .ASCII   /rather than a special terminating character /
      .ASCII   /(<0> or <200>)./<15><12>
```

```
BUFEND:
```

```
.END    START
```

7.19 Accepting a block of characters from the terminal

The following EMT can be used to accept all characters from the terminal input buffer up to and including the last activation character entered. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,115
.WORD     buffer
.WORD     size
```

where "buffer" is the address of the buffer where the characters are to be stored and "size" is the size of the buffer (number of bytes). This EMT causes a program to wait until an activation character is entered and then returns all characters received up to and including the last activation character. On return R0 contains a count of the number of characters received. If the specified buffer overflows, the carry-flag is set on return. This EMT is substantially more efficient than doing a series of .TTYIN EMTs; it is particularly well suited for accepting input from page buffered terminals.

TSX-Plus EMTs

Example:

```
.TITLE  TTIBLK
.ENABL  LC

;Demonstrates the use of TSX-Plus EMT to accept a block of characters
;from a terminal.

.MCALL  .EXIT, .PRINT, .TTYIN

START:  .PRINT  #PROMPT          ;Request input
        MOV     #TTIBLK,R0       ;Point to EMT arg block to
        EMT     375              ;Accept a block of chars from the terminal
        MOV     R0,R1            ;Save input character count
;Char count includes activation char (and LF after CR)
        BCC     1$              ;Buffer overflow on input?
        .PRINT  #OVFLOW         ;Yes, warn user
1$:      ADD     #BUFFER,R1       ;Point past last char in buffer
        CLRB    (R1)             ;Make the input ASCII
        .PRINT  #BUFFER         ;Reproduce the input
        .EXIT

TTIBLK: .BYTE    0,115           ;EMT arg block to accept block from terminal
        .WORD    BUFFER          ;Start of input buffer
        .WORD    <BUFEND-BUFFER> ;Length of buffer in chars (May not exceed
                                ;input buffer size declared in TSGEN.)

        .NLIST  BEX
PROMPT: .ASCIIZ  /70 character input buffer ready./
OVFLOW: .ASCIIZ  /?TTIBLK-F-Buffer overflow/
BUFFER: .ASCII   /XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/ ;35 chars
        .ASCII   /XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/ ;35 chars
BUFEND: .ASCII   /??/           ;TTIBLK will never write over these

.END    START
```

7.20 Program controlled terminal options

Programs may dynamically change various parameters related to terminal control. The following EMT may be used to set various program controlled terminal options:

```
EMT     375
```

with R0 pointing to the following argument block:

```
.BYTE    0,152
.WORD     function-code
.WORD     argument-value
```


where "function-code" is a character which specifies which option is to be set or changed, and "argument-value" specifies a value used only by some options. See the section on program controlled terminal options earlier in this manual for more information on the specific options which may be selected and details on their effects.

Example:

See the example program EMTMTH in the chapter on program controlled terminal options.

7.21 Setting terminal baud rates

The transmit/receive speed for time-sharing lines or CL lines may be set either with the keyboard command SET TT SPEED or from within a program. Line speeds may only be set for terminal interfaces which support programmable baud rates, such as: DLV11-E, DZ(V)11, DH(V)11 type interfaces and the PRO-350 printer and communication ports. The EMT to set line speeds from within a program is:

```
EMT      375
```

with R0 pointing to an argument block of the form:

```
.BYTE    0,152
.WORD    line-number
.WORD    speed-code
```

where "line-number" indicates the TSX-Plus line number for which the speed is to be set, and "speed-code" is a number from 0 to 15 (decimal) indicating the baud rate to be used for both transmit and receive on the line. Split speeds (different transmit and receive baud rates) are not supported. If the line-number is 0, then the speed is set on the line from which the EMT is issued. The speed codes for different baud rates are:

Speed	Code	Speed	Code	Speed	Code
-----	----	-----	----	-----	----
75	1	110	2	134.5	3
150	4	300	5	600	6
1200	7	1800	8	2000	9
2400	10	3600	11	4800	12
7200	13	9600	14	19200	15

Note that the above speed codes are decimal values. A baud rate of 19200 is not supported by DEC DZ(V)11 controllers. DHV11 interfaces do not support 7200 baud. DH11 interfaces do not support 3600 or 7200 baud.

TSX-Plus EMTs

Errors:

<u>Code</u>	<u>Meaning</u>
1	Job does not have operator privilege
2	Invalid line number specified

Example:

```
.TITLE  SETSPD
.ENABL  LC
;
; Demonstrate EMT to set a line's transmit/receive baud rate
;
.MCALL  .PRINT,.EXIT
.GLOBL  PRTDEC
.DSABL  GBL

ERRBYT  = 52                      ;EMT error byte address

START:  MOV    #SETSPD,R0          ;Point to EMT arg block to
      EMT      375                  ;Set line speed
      BCC      9$                  ;Branch if OK
      MOVB     @#ERRBYT,-(SP)       ;Fetch EMT error code
      .PRINT   #ERRIS              ;"EMT error is:"
      MOV      (SP)+,R0             ;Retrieve error code
      CALL     PRTDEC              ;Display it
9$:     .EXIT

SETSPD: .BYTE   0,154              ;EMT arg block to set line speed
      .WORD    6                   ;Line number
      .WORD    14.                ;Speed code for 9600 baud

.NLIST  BEX
ERRIS:  .ASCII  /?SETSPD-F-EMT error code = /<200>
      .EVEN
      .END    START
```

7.22 Assigning a CL unit to a time-sharing line

Time-sharing terminal lines which are not in use may be re-assigned as general purpose serial I/O lines by directing the TSX-Plus CL facility to assign a CL unit to the line. This can be done either with a keyboard command (see the SET CL LINE=n command in Chapter 2) or from within a program. A special system service call (EMT) is available to assign a CL line from within a program. The form of the EMT is:

EMT 375

with R0 pointing to an argument block of the form:

```
.BYTE 0,155
.WORD CL-unit
.WORD line-number
```

where "CL-unit" specifies the CL unit number to be assigned to the line, and "line-number" identifies the TSX-Plus line number to be disabled as a time-sharing line and reassigned as a CL unit. The valid range of CL unit numbers is determined by the number of CL units defined during TSX-Plus system generation. For example, if 3 CL units are defined, then the valid CL units are CL0, CL1 and CL2. If "line-number" is zero, then the CL unit is disassociated from the line and the line is restored to its previous function. The line-number may also refer to lines generated as dedicated CL lines. In this case, when a CL unit is disassociated from the line, it is simply returned to a pool of lines available for CL use, it does not become redefined as a time-sharing line.

See the TSX-Plus System Manager's Guide for more information on CL units.

Errors:

Code	Meaning
----	-----
1	Job issuing request does not have operator privilege
2	Invalid CL unit number specified
3	Invalid line number specified
4	Specified line number already assigned to a CL unit
5	Specified line number in use for time-sharing
6	Specified CL unit is currently busy

Example:

```
.TITLE GETCL
.ENABL LC

;
; Demonstrate EMT to switch time-sharing line to CL line
; Demonstrate EMT to allocate a device
; Demonstrate .SPFUN request to CL
;
; This example attempts to attach a CL unit to
; a line which is linked to another machine,
; allocate it for exclusive use,
; modify the default CL settings,
; and then start up the RT-11 VTCOM utility.
;
; Since it is difficult to make logical assignments from within
; a program, use a special .EXIT to pass the ASSIGN command to
; KMON and then run VTCOM.
```


TSX-Plus EMTs

```

;
;      .MCALL  .PRINT, .EXIT, .LOOKUP, .SPFUN, .PURGE
;      .DSABL  GBL
;
ERRBYT = 52      ;EMT error byte address
JSW     = 44      ;Job Status Word address
CHNIF$  = 4000    ;Chain information bit in JSW
;
START:
;
; Issue EMT to attach the line
;
      MOV      #ATTCL,R0      ;Point to EMT arg block to
      EMT      375            ;Attach CL unit to T-S line
      BCC      1$            ;Branch if OK, else
;
; EMT error, explain it before exiting
;
      MOVB     @#ERRBYT,R1    ;Get EMT error code
      ASL      R1             ;Convert error byte to word index
      .PRINT   ATTERR(R1)     ;Print appropriate error message
      BR       10$           ;And force simple exit
;
; Allocate device so nobody else infringes it
;
1$:    MOV      #ALOCAT,R0     ;Point to EMT arg block to
      EMT      375            ;Allocate CL for exclusive use
      BCC      2$            ;Branch if OK, else
;
; EMT error, explain it before exiting
;
      MOVB     @#ERRBYT,R1    ;Get EMT error code
      ASL      R1             ;Convert to word index
      .PRINT   ALLERR(R1)     ;Print appropriate error message
      BR       10$           ;And exit
;
; Issue SET CL command (not the easiest way to do this).
; If a channel were already open to CL, then this would
; make more sense. But, as an example, why not?
;
2$:    .LOOKUP  #AREA,#0,#CLUNAM      ;Open a channel to CL
      .SPFUN   #AREA,#0,#251,#CLFLAG,#0,#0 ;Turn off flagged options
      .PURGE   #0                  ;Done with channel
;
; Set up for special exit passing commands to KMON
;
      MOV      #COMAND,R1      ;Point to command line text
      MOV      #510,R2        ;Point to chain info area
      MOV      #COMLEN,(R2)+    ;# of bytes in chain data area

```



```

3$:      MOVB      (R1)+,(R2)+      ;Move command lines into chain area
        CMP       R1,#COMEND      ;Reached end yet?
        BLO       3$              ;Repeat if not
        BIS       #CHNIF$,@#JSW    ;Set pass command bit in JSW
                                   ;(aborts any pending command file)
        CLR       R0              ;Required for special exit
        MOV       #1000,SP         ;Reset stack pointer
10$:     .EXIT                    ;Done (Note: unit remains allocated
                                   ;      after program exit!)
;
; EMT arg blocks and word buffers
;
AREA:    .BLKW     10              ;General purpose EMT arg block
CLFLAG:  .WORD     10              ;NOLFOUT flag for CL .SPFUN
;
ATTCL:   .BYTE     0,155           ;EMT arg block to take over TS line by CL
        .WORD     0                ;Selected CL unit number
        .WORD     6                ;Selected TS line number
;
ALOCAT:  .BYTE     0,156           ;EMT arg block to allocate a device
        .WORD     CLUNAM           ;Address of 4-word device specification
CLUNAM:  .RAD50    /CLO/           ;Start with first CL unit
        .WORD     0,0,0           ;Dummy file specification
;
; General messages and byte buffers
;
        .NLIST    BEX
COMAND:   .ASCIZ    /ASSIGN CLO XL/ ;Make logical assignment of XL for VTCOM
        .ASCIZ    /R VTCOM/       ;Run VTCOM (part of RT11 V5.01 kit)
COMEND:   .LIST     BEX
        .EVEN
COMLEN   = COMEND-COMAND           ;# bytes in commands passed to KMOM
        .IF      GT <COMLEN-<1000-512>>
        .ERROR   1                ; Chain data area overflow
        .ENDC
;
ATTERR:  .WORD     0                ;Attach CL EMT error message table
        .WORD     NOPRIV           ; 1
        .WORD     BADCLU           ; 2
        .WORD     BADTSL           ; 3
        .WORD     ALRDCL           ; 4
        .WORD     INUSE            ; 5
        .WORD     CLBUSY           ; 6

```


TSX-Plus EMTs

```

.NLIST BEX
NOPRIV: .ASCIZ /Not privileged to use this program./
BADCLU: .ASCIZ /Attempt to use invalid CL unit./
BADTSL: .ASCIZ /Attempt to use invalid time-sharing line./
ALRDCL: .ASCIZ /Time-sharing line already in use as CL unit./
INUSE: .ASCIZ /Somebody is already using time-sharing line./
CLBUSY: .ASCIZ /CL unit already active./
.LIST BEX
.EVEN

;
ALLERR: .WORD 0 ;Allocate EMT error table
        .WORD ALRDAL ; 1
        .WORD BADDEV ; 2
        .WORD ALLFUL ; 3
        .WORD DEVUSE ; 4
.NLIST BEX
ALRDAL: .ASCIZ /CL unit already allocated by someone else./
BADDEV: .ASCIZ /Cannot allocate that device./
ALLFUL: .ASCIZ /Too many devices already allocated./
DEVUSE: .ASCIZ /Device in use by someone else./
.LIST BEX
.EVEN

;
.END START

```

7.23 Allocating a device for exclusive use

Devices may be allocated for exclusive use by a single user. This prevents mixing input and output on common, but non-spoiled, devices like a communications line (XL or CL devices) or a magnetic tape. Access restriction by device allocation remains in effect until deallocated by the job or until the job logs off. See the description of the ALLOCATE command in Chapter 2 for more information on device allocation.

There are three system service calls relating to device allocation: allocate a device; deallocate a device; check to see if a device is allocated by another user. The form of the EMT is the same for all three:

EMT 375

with R0 pointing to an argument block of the form:

```

.BYTE n,156
.WORD device-pointer

```

where "device-pointer" is the address of a four word block in which the first word contains the RAD50 name of the device to be allocated and the next three words contain zeros. The specific function is defined by the first byte of the argument block, as follows:

n	Function
0	Allocate a device
1	Deallocate a device
2	Check to see if a device is allocated by another user

You can only allocate a device if no other user has already allocated the device or has a channel open to it. If the device is allocated to another job or another job has a channel open to the device, then the number of the job which is accessing the device is returned in R0. If a job which has already allocated the device or has a channel open to the device is not associated with the same primary line as the job attempting to allocate the device, then the carry bit will be set on return from the EMT and the number of the job which is accessing the device will be returned in R0. If the device is currently allocated by a job which was started from the same primary line as the job which is now attempting to allocate it, then the carry flag will be clear on return, but the other job number will be returned in R0. If both the job with the current allocation and the job attempting to allocate the device are virtual lines, the first one to allocate the device gets the allocation. If either is the primary line, then the primary line will get the allocation.

Errors:

Code	Meaning
1	Device is already allocated by another job
2	Invalid device specified
3	Device allocation table is full (TSGEN parameter MAXALC)
4	Device is currently in use by another job

Example:

See the example program GETCL in the section on assigning a CL unit to a time-sharing line.

7.24 Turning high-efficiency terminal mode on and off

TSX-Plus offers a "high-efficiency" mode of terminal operation that eliminates a substantial amount of system overhead for terminal character processing by reducing the amount of processing that is done on each character. When in high-efficiency mode, characters are sent directly to the terminal with minimum handling by TSX-Plus; operations such as expanding tabs to spaces and form-feeds to line-feeds are omitted as well as input processing such as echoing characters and recognizing control characters such as DELETE, control-U and control-C. The only characters treated specially on input are user-defined activation characters and the user-specified break character. At least one user specified activation character must be declared if high-efficiency mode is to be used. This form of terminal I/O is designed to facilitate high-speed machine-to-machine communication. It can be used effectively to communicate with buffered mode terminals. The form of the EMT used to control high-efficiency mode is:

TSX-Plus EMTs

EMT 375

with R0 pointing to the following argument block:

.BYTE code,120

where "code" is 1 to turn high-efficiency mode on and 0 to turn it off.

Example:

.TITLE HIEFF
.ENABL LC

;Demonstrate the use of TSX-Plus Hi-efficiency terminal mode

```

.MCALL .EXIT,.PRINT
START: .PRINT #DCLCC           ;Make ^C an activation char
       .PRINT #PROMPT         ;Ask for input
       MOV    #HIEFF,R0       ;Point to EMT arg block to
       EMT    375              ;Turn on hi-efficiency mode
       MOV    #TTIBLK,R0      ;Point to EMT arg block to
       EMT    375              ;Accept a block of characters
                                   ;Actual character count returned in R0
; . . .                          Do something useful with the input?
       MOVB   #15,<BUFFER-1>(R0) ;Replace the activation char with
       MOVB   #12,BUFFER(R0)    ;Carriage return, line feed
       INC    R0                ;Count LF for output
       MOV    R0,<TTOBLK+4>      ;Set up count for output
       MOV    #TTOBLK,R0        ;Point to EMT arg block to
       EMT    375              ;Display a block of characters
       CLRB   HIEFF            ;Get ready to turn hi-eff off
       MOV    #HIEFF,R0        ;Point to EMT arg block to
       EMT    375              ;Turn off hi-efficiency mode

.EXIT

HIEFF: .BYTE   1,120           ;EMT arg block to turn hi-eff mode on (off)
TTIBLK: .BYTE  0,115           ;EMT arg block to accept a block of chars
        .WORD  BUFFER          ;Pointer to input buffer
        .WORD  BUFSIZ          ;Number of chars to input
TTOBLK: .BYTE  0,114           ;EMT arg block to display a block of chars
        .WORD  BUFFER          ;Pointer to buffer for output
        .WORD  BUFSIZ          ;Size of buffer to output
BUFFER: .BLKB  82.             ;I/O buffer - Cannot exceed line's I/O
BUFSIZ = . - BUFFER           ;      buffer sizes declared in TSGEN
        .WORD  0               ;Spacer in case of buffer overflow
        .NLIST BEX

DCLCC: .ASCII  <35><^D><3><200> ;Declare ^C as special activation char
PROMPT: .ASCII  /Please enter 1 line of characters (^C ends)./<15><12>
        .ASCIIZ /No special processing or echoing will be done./

```



```
.END    START
```

7.25 Determining number of free blocks in spool file

The following EMT will return in R0 the number of free blocks in the spool file. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    0,107
```

Example:

```
.TITLE   SPLFRE
.ENABL   LC
```

;Demonstrate EMT to determine number of free spool blocks

```
.MCALL   .PRINT,.EXIT
.GLOBL   PRTDEC
```

```
START:   .PRINT   #NUMFRE      ;Preface number message
          MOV      #SPLFRE,R0  ;Point to EMT arg block to
          EMT      375         ;Determine number of free spool blocks
                                   ;Number is returned in R0
          CALL     PRTDEC      ;Display the number
          .PRINT   #BLOKS      ;End of message
          .EXIT
          .NLIST   BEX

SPLFRE:   .BYTE    0,107       ;EMT arg to get # free spool blocks
NUMFRE:   .ASCII   /The spool file has /<200>
BLOKS:    .ASCIZ   / free blocks./
          .EVEN
```

```
.END     START
```

7.26 Set/Reset ODT activation mode

The following EMT can be used to set TSX-Plus to activate on characters that are appropriate to ODT. In this mode TSX-Plus considers all characters to be activation characters except digits, `', `\$', and `;'. The form of the EMT is:

TSX-Plus EMTs

EMT 375

with R0 pointing to the following argument area:

.BYTE code,111

where "code" = 1 to turn on ODT activation mode, and "code" = 0 to reset to normal mode.

Example:

.TITLE ACTODT
.ENABL LC

;Demonstrate EMT which sets ODT activation mode

.MCALL .PRINT,.EXIT

```
START: .PRINT #ODTTYP      ;Say we are entering ODT activation mode
        MOV    #ACTODT,R0  ;Point to EMT arg block to
        EMT    375         ;Set ODT activation mode
1$:      CALL   GETLIN      ;Get some terminal input
        CMPB   BUFFER,#~Q  ;Back to regular mode?
        BNE    1$          ;No, get more lines
        .PRINT #REGTYP     ;Say we are going back to regular activation
        CLRB   ACTODT      ;Make arg block into RESET mode request
        MOV    #ACTODT,R0  ;Point to EMT arg block to
        EMT    375         ;Reset ODT activation mode
2$:      CALL   GETLIN      ;Get more input
        CMPB   BUFFER,#~Q  ;Want to quit?
        BNE    2$          ;No, repeat
```

.EXIT

```
GETLIN: .PRINT #PROMPT     ;Request some input
        MOV    #TTIBLK,R0  ;Point to EMT arg block to
        EMT    375         ;Accept a block of characters
        CLRB   BUFFER(R0)  ;Make input string ASCII
        .PRINT #BUFFER     ;And echo same string back
        RETURN
```

```
ACTODT: .BYTE 1,111        ;EMT arg block to SET/RESET ODT act'n mode
TTIBLK: .BYTE 0,115        ;EMT arg block to get block input from term
        .WORD  BUFFER      ;Pointer to input buffer
        .WORD  79.         ;Number of input chars requested
        .NLIST BEX
ODTTYP: .ASCIZ /Starting ODT activation mode./
REGTYP: .ASCIZ /Restoring regular activation mode./
PROMPT: .ASCII /?/<200>
        .EVEN
```



```

BUFFER: .BLKB  79.           ;TTIBLK input buffer
        .BYTE  0             ;CLRB could go here on full buffer
        .END    START

```

7.27 Determining file directory information

This EMT returns directory information about a file. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

.BYTE  chan,145
.WORD  dblk
.WORD  rblk

```

where "chan" is a channel number in the range 0-16 (octal) that is currently not in use, "dblk" is the address of a 4-word block containing the RAD50 file specification (device, file name, extension), and "rblk" is the address of a 7-word block that will receive the information about the file. The information returned in "rblk" is:

```

Word 1:  Size of the file (number of blocks).
Word 2:  0-->File not protected; 1-->File is protected.
Word 3:  File creation date (standard RT-11 date format).
Word 4:  File creation time (number of 3-second units).
Word 5:  Starting block number of file.
Word 6:  Unused (reserved)
Word 7:  Unused (reserved)

```

Errors:

Code	Meaning
0	Channel is currently in use.
1	Unable to locate specified file.
2	Specified device is not file structured.

Example:

```

.TITLE  FILINF
.ENABL  LC

```

; Demonstrate TSX-Plus EMT to return information about a file

```

.MCALL  .PRINT,.EXIT,.CSISPC,.TTYOUT
.GLOBL  DSPDAT,DSPTI3,PRTDEC

```

```
ERRBYT  = 52           ;EMT error code location
```

```
START:  .CSISPC #OUTSPC,#DEFLT,#0,#BUFFER      ;Get file name
```


TSX-Plus EMTs

```

MOV      #FILINF,R0      ;Point to EMT arg block to
EMT       375             ;Get information about a file
BCC       5$             ;No error?
MOVB     @#ERRBYT,R1      ;What error
ASL       R1             ;Convert to word index
.PRINT    FIERR(R1)       ;Explain it
.EXIT

5$: MOV     #BUFFER,R0    ;Find the end of the file spec.
10$: TSTB   (R0)+         ;End?
      BNE    10$          ;No, keep looking
      MOVB   #200,-(R0)    ;No CR,LF at end
      .PRINT #BUFFER      ;File name
      .TTYOUT #~[
      MOV     FILSIZ,R0    ;File size
      CALL    PRTDEC
      TST     PRTCTD       ;Was file protected?
      BEQ     15$
      .TTYOUT #~P
15$: .TTYOUT #~]
      .PRINT  #SPACE2
      MOV     FILDAT,R0    ;File creation date
      CALL    DSPDAT       ;Display date
      .PRINT  #SPACE2
      MOV     FILTIM,R0    ;File creation time (3 sec resolution)
      CALL    DSPTI3       ;Display special 3-sec time
      .PRINT  #SPACE2
      MOV     FILLOC,R0    ;File starting block #
      CALL    PRTDEC
      .EXIT

.NLIST    BEX
FILINF: .BYTE 0,145      ;EMT arg block to get file info.
      .WORD  INSPC        ;Pointer to RAD50 file name
      .WORD  FILSIZ       ;Pointer to 7 word result buffer
FILSIZ: .WORD 0          ;File size
PRTCTD: .WORD 0          ;Protected=1, unprotected=0
FILDAT: .WORD 0          ;File date (standard format)
FILTIM: .WORD 0          ;File time (special 3-sec format)
FILLOC: .WORD 0          ;File starting block number
      .WORD  0,0          ;Pad for 2 reserved words
OUTSPC: .BLKW 15.        ;Output file specifications
INSPC: .BLKW 24.         ;Input file specifications
DEFLT: .WORD 0,0,0,0     ;No default extensions
FIERR: .WORD CINUSE      ;EMT error message table
      .WORD  NOFILE
      .WORD  BADDEV
BUFFER: .BLKB 81.        ;Input string buffer
SPACE2: .ASCII / /<200>
CINUSE: .ASCIZ /?FILINF-F-Channel in use./

```



```

NOFILE: .ASCIZ  /?FILINF-F-Can't find file./
BADDEV: .ASCIZ  /?FILINF-F-Non-directory device./
        .END    START

```

7.28 Setting file creation time

The time that a file is created is stored along with other directory information under TSX-Plus. In order to pack the time into a single word, TSX-Plus represents the file creation time in three second units. For example, if a file was created at 11:13:22, then the special time representation would be 13467 (decimal).

```

11 hr * 60 min/hr * 60 sec/min = 39600
   13 min      * 60 sec/min =   780
        22 sec      =       22
                        -----
                        40402 seconds

```

40402 sec / 3 sec/unit = 13467 3-sec units

A utility program is provided with TSX-Plus to display the creation time and other directory information about a file. See Appendix C for more information on the FILTIM utility.

The creation date and time for a file are automatically stored by TSX-Plus in the directory entry for the file at the time that the file is closed after being created. An EMT is provided for those unusual situations where a different creation time is to be specified for a file after the file is created. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

.BYTE  chan,146
.WORD  dblk
.WORD  time

```

where "chan" is the number of an unused channel, "dblk" is the address of a 4-word block containing the RAD50 file specification, and "time" is the time value (in 3-second units since midnight) that is to be set as the creation time for the file.

Errors:

<u>Code</u>	<u>Meaning</u>
0	Channel is currently in use.
1	Unable to locate specified file.
2	Specified device is not file structured.

TSX-Plus EMTs

Example:

```

        .TITLE   SFTIM
        .ENABL   LC

; Demonstrate TSX-Plus EMT to set file creation time

        .MCALL   .PRINT,.CSISPC,.EXIT,.GTLIN
        .GLOBL   ACRTI3                ;Subroutine to convert hh:mm:ss to special
                                         ;3-sec internal time format in R0

ERRBYT  = 52                          ;EMT error code location

START:   .PRINT  #GETNAM                ;Prompt for file name
        .CSISPC  #OUTSPC,#DEFLT         ;Get file name in RAD50
        .GTLIN   #BUFFER,#GETTIM       ;Prompt for and get a time
        MOV      #BUFFER,R0            ;Point to time input buffer
        CALL     ACRTI3                 ;Get special time in R0
        BCC      1$                    ;Time error?
        .PRINT   #BADTIM                ;Yes, incorrect format
        .EXIT

1$:      MOV      R0,NEWTIM              ;Save special time
        MOV      #SFTIM,R0              ;Point to EMT arg block to
        EMT      375                     ;Set creation time in file
        BCC      2$                    ;Error?
        MOVB     @#ERRBYT,R0            ;Yes, get error code
        ASL      R0                      ;Convert to word offset
        .PRINT   SFTERR(R0)             ;Explain

2$:      .EXIT

        .NLIST   BEX

SFTIM:   .BYTE    0,146                 ;EMT arg block to set file creation time
        .WORD     INSPC                 ;Pointer to RAD50 file name
NEWTIM:  .WORD     0                     ;Will contain new creation time
OUTSPC:  .BLKW     15.                   ;.CSISPC output files
INSPC:   .BLKW     24.                   ;.CSISPC input files (first is the one)
DEFLT:   .WORD     0,0,0,0               ;Default file extensions
SFTERR:  .WORD     INUSE                 ;SFTIM error message table
        .WORD     NOFILE
        .WORD     BADDEV

BUFFER:  .BLKB     81.                  ;.GTLIN input buffer - holds time hh:mm:ss
GETNAM:  .ASCII    /Set creation time in file: /<200>
GETTIM:  .ASCII    /New creation time: /<200>
BADTIM:  .ASCIZ    /?SFTIM-F-Invalid time./
INUSE:   .ASCIZ    /?SFTIM-F-Channel in use./
NOFILE:  .ASCIZ    /?SFTIM-F-Can't find file./
BADDEV:  .ASCIZ    /?SFTIM-F-Non-directory device./
        .END      START

```


8. TSX-Plus JOB ENVIRONMENT

8.1 Virtual and physical memory

The memory space that is accessible by a job is known as the virtual address space for the job. Because of the architectural design of the PDP-11 computer which uses 16 bits to represent a virtual memory address, the maximum amount of virtual address space that can be accessed at one time by a job is limited to 65,536 (64K) bytes. Thus, the virtual addresses for a job range from 000000 to 177777 (octal).

The actual amount of virtual address space available to a job may be as large as 64Kb but it may be restricted to less than this amount. The following factors control the size of the virtual address space available to a job:

- 1) The maximum amount of memory allowed for each job as determined by the HIMEM system generation parameter.
- 2) The amount of memory specified with the MEMORY keyboard command (initialized by the DFLMEM system generation parameter).
- 3) The memory limit reserved in the disk file image by the SETSIZ program (see Appendix A).
- 4) The amount of memory acquired by use of the TSX-Plus EMT that expands or contracts the job space.

The physical address space for a PDP-11 computer is not limited to 64Kb. The maximum physical address space depends on the model of PDP-11 and the amount of memory installed on the computer. LSI-11/23 and 11/34 computers can access up to 256Kb of physical memory. The 11/23-Plus, 11/73, 11/24 and 11/44 computers can access up to 4Mb of memory.

The process by which an address in the job's virtual address space is transformed into an address in the physical address space is known as mapping. The mapping of the virtual address space for a job into the physical memory space assigned to the job is performed by the memory management hardware facility of the PDP-11 computer. This facility divides the virtual address space into 8 sections, called pages, each of which can address up to 8Kb of memory. The mapping of a page of virtual address space to a page of physical address space is accomplished by setting up information in a page address register (PAR). There is one page address register for each of the 8 virtual address pages. These registers are not directly accessible by a user job but are loaded by the TSX-Plus system when it starts a program, changes the size of a program, or switches execution between different jobs. The relationship between the 8 pages of memory and the corresponding sections of virtual address is shown in the following table:

TSX-Plus Environment

Page	Virtual address range
0	000000 - 017777
1	020000 - 037777
2	040000 - 057777
3	060000 - 077777
4	100000 - 117777
5	120000 - 137777
6	140000 - 157777
7	160000 - 177777

Because of the design of the memory management system in the PDP-11, it is not possible to divide the virtual address space more finely than 8 pages of 8Kb each. However, it is possible to map each page of virtual address space into any section of physical memory. (This facility allows TSX-Plus to keep multiple user jobs in physical memory and to switch rapidly among them by reloading the page address registers.)

8.2 User virtual address mapping

The virtual address space accessed by a job can be divided into five categories:

- 1) Normal program space which is used by instructions and data for programs.
- 2) Simulated RMON. This is the virtual address region from 160000 to 177777 that is mapped to a simulated RMON (RT-11 resident monitor).
- 3) Extended memory windows. Programs can create regions in physical memory and then cause one or more pages of virtual address space to be mapped to the regions.
- 4) Shared run-time systems. Several TSX-Plus jobs can cause a portion of their virtual address space to be mapped to the same area of physical memory. This allows several users to execute the same program or share common data without having to allocate a separate area of physical memory for each user.
- 5) System I/O page. TSX-Plus real-time programs may map the system I/O page into their virtual address space.

These categories of virtual address space are discussed in the following sections.

8.3 Normal programs and virtual programs

Programs run under TSX-Plus may be divided into two categories: normal programs and virtual programs. The only difference between the two types of programs is the manner in which TSX-Plus handles page 7 (addresses 160000 to 177777) of the virtual address space. In the case of normal programs, page 7 is mapped to a simulated RMON. RMON is the name of the resident RT-11 monitor. When running under RT-11 this is the actual system control program. When running under TSX-Plus, the simulated RMON does not contain any of the instructions that are part of RT-11 but contains only a table that provides information about the system and the job. This information includes such items as the system version number, and information about the hardware configuration. The cells in this table are known as RMON fixed offsets. Their position within the table and their contents are documented in the RT-11 Software Support Manual, although not all cells are relevant to or maintained by TSX-Plus.

The address of the base of the simulated RMON table is stored in location 54 of the job's virtual address space. Modern RT-11 and TSX-Plus programs should not directly access the RMON table but rather should use the .GVAL EMT to obtain values from the table. However, since some older programs and some RT-11 utility programs directly access the RMON tables, it is mapped through page 7 for normal programs. As a result, normal programs are restricted to using pages 0 to 6 (56Kb) for their own instructions and data.

Virtual programs are programs that do not require direct access to the simulated RMON table. These programs may still access the RMON values with the .GVAL and .PVAL EMTs. Since direct access to the simulated RMON is not needed, page 7 is available for the program to use for its own instructions and data, thus providing a total of 64Kb of virtual address space. A program may indicate that it is a virtual program by any of the following techniques:

- 1) Set bit 10 (VIRT\$ -- mask 2000) in the Job Status Word (location 44) of the SAV file. See Appendix A for information about how this bit can be set by use of the SETSIZ program.
- 2) Use the /V LINK switch (/XM switch for the LINK keyboard command) which stores the RAD50 value for "VIR" in location 0 of the SAV file.
- 3) Use the TSX-Plus SETSIZ program (see Appendix A) and indicate that more than 56Kb of memory is to be used for the program.

8.4 Extended memory regions

Programs running under TSX-Plus have available the Programmed Logical Address Space (PLAS) facility that is provided by the RT-11XM monitor. This facility allows a program to allocate regions of physical memory and then create virtual windows that can be used to access the regions. There are 7 system service calls (EMTs) provided for PLAS support:

TSX-Plus Environment

- .CRRG -- Create a region
- .ELRG -- Eliminate a region
- .CRAW -- Create a virtual address window
- .ELAW -- Eliminate a virtual address window
- .MAP -- Map a virtual window to a region
- .UNMAP -- Unmap a virtual window
- .GMCX -- Get information about the status of a window

A region is an area of physical memory set aside for use by a job in addition to its normal job space. The .CRRG EMT is used by a program to request that a region be created. The size of a region is not restricted to 64Kb and may be as large as the physical memory installed on the system (less the space used by the TSX-Plus system, device handlers, tables, and the remainder of the program). Up to 8 regions may be created by each job.

In order to access a region, a program must use the .CRAW and .MAP EMTs to create a virtual window and map the virtual window to a selected portion of the region. A virtual window is a section of virtual address space mapped to a region rather than to the normal job physical address space. Up to 8 virtual address windows can be created by each job. The same virtual window (i.e., the same range of virtual addresses) may be mapped to different regions or different sections of the same region at different times by use of the .MAP EMT. This allows a program to selectively access different sections of code or data in extended memory regions during the course of its execution.

When an extended memory region is created by use of the .CRRG EMT, space is allocated in physical memory and in a TSX-Plus region swap file. Whenever a job is swapped out of memory, its extended memory regions are swapped to the region swap file. Space in the region swap file is allocated and deallocated dynamically as regions are created and eliminated. In order to create a region, space must be available in physical memory and in the region swap file.

The PLAS facility is most commonly used implicitly through the virtual overlay and virtual array features. Using the PLAS facilities, it is possible for a single job to use all of the physical memory space available on a system (exclusive of the space used by the TSX-Plus system, handlers, tables, etc.). Proper use of the PLAS facilities such as with reasonable size virtual overlays or arrays can lead to substantial performance improvements for programs. Excessive use of memory space with the PLAS facility can lead to excessive job swapping and degraded system performance.

8.5 Shared run-time systems

A shared run-time system is a program or data area in physical memory that can be accessed by multiple TSX-Plus jobs. Shared run-time systems are somewhat similar to extended memory regions in that they are both allocated in extended memory areas and must be accessed by mapping a portion of the jobs virtual address space to the physical memory area. The difference is that extended memory regions are private to the job that creates them and may not be accessed by any other job. Shared run-time systems can be simultaneously accessed

(hence "shared") by any number of TSX-Plus jobs. Another difference between regions and shared run-time systems is that regions can be created dynamically and can be swapped out of memory; shared run-time systems are specified when the system is generated and reside in memory as long as the system is running. See Chapter 12 for more information on shared run-time systems.

8.6 Access to system I/O page

The system I/O page is an 8Kb section of addresses which is not connected with ordinary memory but rather is used to control peripheral devices and hardware operation. Access to the I/O page is risky in that a program can interfere with peripheral devices and cause system crashes. For this reason, programs do not ordinarily have access to the I/O page. However, a program that is running with TSX-Plus real-time privilege may issue a system service call to cause page 7 (160000-177777) of the job's virtual address space to be mapped to the system I/O page. See Chapter 11 for more information on real-time programs.

8.7 VM pseudo-device handler

While the VM handler is not actually mapped into a job's memory space, its use can dramatically increase job performance. The VM handler enables the use of a portion of physical memory as a pseudo-disk device. This permits very rapid access to programs and data which are placed on the VM unit. For programs such as compilers which heavily utilize overlay segments, a considerable speed-up can be achieved by loading them onto the VM device. A similar improvement for overlaid programs can also be obtained with the general data cache facility. However, when the data cache is full the least recently used blocks are lost. The presence of particular programs and their overlay segments in memory can be guaranteed by copying them to the VM pseudo-device. Another example of the usefulness of the VM device is with compilers which heavily use temporary work files. Depending on the number of write operations, which are not helped by general data caching, significant improvements in speed can be obtained by directing the work files to the VM pseudo-device.

In order to use the VM device, it must be included in the device definitions during TSX-Plus system generation. An upper limit must also be placed on the amount of memory available to TSX-Plus. The physical memory above that available to TSX-Plus can then be used as a memory based pseudo-disk. If for some reason, it is desirable to use less than all of the memory above the top of TSX-Plus, the SET VM BASE command can be used to restrict the memory available to VM. Each time TSX-Plus is restarted, VM must be initialized just as you would for a new physical disk or a fresh logical subset disk. For example:

INITIALIZE VM:

Only one unit (VM0:) is available; however, logical subset disks may be created within the VM pseudo-device to partition it if necessary. On initialization, the VM handler automatically determines the amount of memory available to it.

TSX-Plus Environment

See the TSX-Plus System Manager's Guide for more information on the use of data caching (general and shared files) and the VM pseudo-disk.

9. SHARED FILE RECORD LOCKING

TSX-Plus allows several programs to have the same file open simultaneously. In order to control access to such files, TSX-Plus provides system calls to "lock" shared files and records within shared files. Through the record locking facility a program may gain exclusive access to one or more blocks in a file by locking those blocks. Other users attempting to lock the same blocks will be denied access until the first user releases the locked blocks. The TSX-Plus shared file facility also provides data caching on blocks being read from shared files.

Note that shared file access protection is only meaningful for cooperating jobs requesting shared access. This scheme does not prevent other jobs from opening or writing to files if those jobs do not adhere to the file sharing protocol.

The usual protocol for updating a shared file being accessed by several users is as follows.

- 1) Open file.
- 2) Tell TSX-Plus that file is "shared".
- 3) Lock all blocks in file which contain desired record.
- 4) Read locked blocks into memory.
- 5) Make update to record.
- 6) Write updated blocks to file.
- 7) Unlock blocks.
- 8) Repeat steps 3-7 as needed.
- 9) Close file.

DIBOL record locking procedures

Subroutines to control record locking from within DIBOL programs are provided with TSX-Plus. These are discussed in Appendix B.

Record locking from other languages

Record locking may be interfaced to other languages with appropriate subroutine calls. Record locking under COBOL-Plus is built into the run-time library provided with COBOL-Plus. The remainder of this chapter describes the techniques used to control shared file access and record locking.

9.1 Opening a shared file

Before a file can be used with shared access it must be opened by using a standard .LOOKUP EMT. After the file has been successfully opened, the following EMT may be used to declare the file to be opened for shared access. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    chan,125  
.WORD    access-code
```

where "chan" is the number of the I/O channel open to the desired file and "access-code" is a value indicating the type of access protection desired for the file. The following access codes are recognized:

Shared File Record Locking

Code	Protection	Access
0	Exclusive	Input
1	Exclusive	Update
2	Protected	Input
3	Protected	Update
4	Shared	Input
5	Shared	Update

The access-code specifies two things: The type of access that you intend to make to the file (input only or update) and the type of access that you are willing to grant to other users of the file. There are three protection classes: Exclusive, Protected and Shared. Exclusive access means that you demand exclusive access to the file and will allow no other users to access the file in any fashion (input or update). Protected access means that you will allow other users to open the file for input but wish to prohibit any other users from opening the file for update. Shared access means that you are willing to allow other users to open the file for both input and update access.

When this EMT is executed, TSX-Plus checks your specified protection mode and access type with that previously declared for the file by other users. If an access conflict arises because of your specified access characteristics an error code of 4 is returned for the EMT. If no access conflict is detected, your specified access code is saved with the file and will be used to check for conflicts with future shared access requests issued by other users.

Normally all files that are declared to TSX-Plus using this EMT are enabled for use of the data caching facility (see description below). However, in some cases it may be desirable to suppress data caching for certain files. For example, sequential access files usually benefit little from data caching and enabling data caching for these files causes the data cache buffers to be used non-productively when they could be providing a better service for other types of files. To disable data caching for a file set bit 8 (octal 400) in the access-code word. When shared access is declared with bit 8 set, new data is not brought into the data cache when the file is read. However, if the data being read is already stored in the cache because of a read by another user, it is used. When data being written to a file is currently stored in the cache, the data in the cache is updated even if the file is declared to be non-cached.

It is possible to have several channels simultaneously open to different shared files. The exact number of channels that can be open to shared files and the total number of shared files that may be opened are specified when the TSX-Plus system is generated.

Once all access to a shared file is completed, the I/O channel should be closed using the standard .CLOSE or .PURGE EMT's. See the next section for information about saving the status of a channel that has been opened to a shared file.

The error codes that can be returned by this EMT are listed below:

Errors:

Code	Meaning
1	Channel has not been opened to a file.
2	Too many channels opened to shared files.
3	Too many shared files open.
4	File protection-access conflict

Example:

```
.TITLE  SHARED
.ENABL  LC
```

; This program cooperates with the example program (SHARE2) in the
; following section to demonstrate shared file access protection.

```
.MCALL  .PRINT, .GTLIN, .TWAIT, .EXIT, .READW
.MCALL  .LOOKUP, .CLOSE, .SAVESTATUS, .REOPEN, .PURGE
```

```
ERRBYT = 52                ;EMT error byte
EXUP = 1.                  ;Shared file access code: Exclusive, Update
PRIN = 2.                  ;Shared file access code: Protected, Input
BUFSIZ = 256.              ;Number of words in a disk block

START:  .LOOKUP #AREA, #0, #SHR1 ;Open SHR1.DAT
        BCC     1$             ;Branch if OK
        .PRINT  #LKPERR        ;Lookup error message
        .EXIT

1$:      MOV     #EXUP, <SHRFIL+2> ;Set Exclusive, Update access
        MOV     #SHRFIL, R0      ;Point to EMT arg block to
        EMT     375              ;Declare SHR1.DAT as a shared file
                                   ;with Exclusive and Update access
        BCC     2$             ;Branch if sharing OK
        JMP     EMterr          ;Explain the error and quit

2$:      .READW  #AREA, #0, #BUFFER, #BUFSIZ, #0 ;Read block 0 of SHR1.DAT
        BCC     3$             ;Branch if read OK
        .PRINT  #RDWERR        ;Say there was a read error
        .EXIT

3$:      .PRINT  #BUFFER        ;Print out the file (must have 0 or 200 byte)
        .SAVESTATUS #AREA, #0, #BLOK1 ;Save channel 0 status for reuse
        BCC     4$             ;Branch if savestatus OK
        .PRINT  #SVSERR        ;Savestatus error message
        .EXIT

4$:      MOV     #SAVSHR, R0     ;Point to EMT arg block to
        EMT     375              ;Save shared file status
        .PURGE  #0              ;Purge the channel for reuse
        .LOOKUP #AREA, #0, #SHR2 ;Open SHR2.DAT
        BCC     5$             ;Branch if OK
```


Shared File Record Locking

```

        .PRINT  #LKPER2          ;Say bad lookup on SHR2
        .EXIT
5$:      MOV     #PRIN,<SHRFIL+2> ;Set Shared, Input access
        MOV     #SHRFIL,R0      ;Point to EMT arg block to
        EMT     375             ;Declare SHR2.DAT as a shared file
        BCC     6$              ;Branch on no error
        JMP     EMTER2          ;Say error on SHR2 sharing
6$:      .READW  #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Read block 0 of SHR2.DAT
        BCC     7$              ;Branch if read OK
        .PRINT  #RDWER2          ;Say read error on SHR2
        .EXIT
7$:      .PRINT  #BUFFER          ;Print out the contents (1 line, null filled)
        .PRINT  #PROMPT          ;Say it's time to try companion program
        .WAIT   #AREA,#TIME      ;Wait 30 seconds to run other program
        .PURGE  #0              ;Now, release SHR2
        .GTLIN  #BUFFER,#PRMPT2 ;Wait for return from virtual line
;This job will be suspended for output while gone to virtual line
        .REOPEN #AREA,#0,#BLOK1 ;And get SHR1 back
        .READW  #AREA,#0,#BUFFER,#BUFSIZ,#1 ;Read in second block of SHR1
        .PRINT  #BUFFER          ;And print it to prove status was saved
        .CLOSE  #0              ;Release SHR1
        .EXIT
EMTER2:  MOV     #SHR2NM,FILNUM   ;Point to alternate file error
EMTERR:  MOV     @#ERRBYT,R0      ;Get the error type
        DEC     R0              ;Zero offset
        ASL     R0              ;Convert to word offset
        .PRINT  SHRERR(R0)       ;Print the appropriate error message
        .PRINT  FILNUM          ;And the file name
        .EXIT

AREA:    .BLKW   10              ;EMT arg block area
BLOK1:   .BLKW   5               ;Savestatus area for SHR1.DAT
FILNUM:  .WORD   SHR1NM          ;File name for error message
TIME:    .WORD   0,30.*60.       ;30.sec * 60.tics/sec
SHRERR:  .WORD   NOTOPN          ;Pointer to EMT error messages
        .WORD   XSSCHN
        .WORD   XSSFIL
        .WORD   AXSCON
        .NLIST  BEX

SHR1:    .RAD50  /DK SHR1  DAT/   ;File descriptor for SHR1.DAT
SHR2:    .RAD50  /DK SHR2  DAT/   ;File descriptor for SHR2.DAT
SHRFIL:  .BYTE   0,125           ;EMT arg block to declare shared file
        .WORD   1               ;Exclusive Update access (GETS CHANGED)
SAVSHR:  .BYTE   0,122           ;EMT arg block to save shared file status
NOTOPN:  .ASCII  /Attempt to share unopened channel/<7><200>
XSSCHN:  .ASCII  /Too many channels opened to shared files/<7><200>
XSSFIL:  .ASCII  /Too many shared files open/<7><200>
AXSCON:  .ASCII  /Attempt to protect already protected shared file/<7><200>
SHR1NM:  .ASCIZ  /: SHR1.DAT/
SHR2NM:  .ASCIZ  /: SHR2.DAT/

```



```
LKPERR: .ASCIZ  /Lookup error for SHR1.DAT/<7>
LKPER2: .ASCIZ  /Lookup error for SHR2.DAT/<7>
SVSERR: .ASCIZ  /Error occurred attempting to save SHR1.DAT file status/<7>
RDWERR: .ASCIZ  /Error occurred while reading SHR1.DAT/<7>
RDWER2: .ASCIZ  /Error occurred while reading SHR2.DAT/<7>
PROMPT: .ASCII  /Go to a virtual line and RUN SHARE2 which attempts /<15><12>
        .ASCII  /to share the same files (SHR1.DAT, SHR2.DAT)./<15><12>
        .ASCIZ  /Waiting 30 seconds . . . . ./<7>
PRMPT2: .ASCII  /When you have returned, hit RETURN to continue/<200>
BUFFER: .BLKW   BUFSIZ
```

.END START

See also the example program SHARE2 in the section on saving the status of a shared file channel.

9.2 Saving the status of a shared file channel

A standard .SAVESTATUS EMT may be used to save the status of a shared file channel. If this is done, all blocks that are being held locked in the file remain locked until the channel is reopened and the blocks are unlocked (see below).

When using a single channel number to access several shared files it is convenient to initially do a .LOOKUP on each file, then declare the file to be shared (EMT above), and then do a .SAVESTATUS. The channel being used to access the set of files can then be switched from one file to another by doing a .PURGE followed by a .REOPEN. However, before doing the .PURGE, TSX-Plus must be told that you wish to save the shared-file status of the file, otherwise all locked blocks will be unlocked and the file will be removed from the shared-file list. The form of the EMT used to perform this function is:

```
EMT       375
```

with R0 pointing to the following argument block:

```
.BYTE    chan,122
```

where "chan" is the I/O channel number. The effect of this EMT is to suspend the connection between the shared file information table and the I/O channel. Any blocks that are currently locked in the file remain locked until the channel is reopened to the file (by using a standard .REOPEN EMT). After saving a shared file status, the channel may be freed by using a .PURGE EMT.

Shared File Record Locking

Example:

```
.TITLE  SHARE2
.ENABL  LC
```

```
; This program cooperates with the example program (SHARED) in the
; previous section to demonstrate saving of shared file status with
; the .SAVSTATUS EMT.
```

```
.MCALL  .LOOKUP,.PRINT,.EXIT,.READW
.MCALL  .CLOSE,.TWAIT
```

```
ERRBYT = 52                ;EMT error byte
BUFSIZ = 256.              ;Size of disk file block
```

```
START:  .LOOKUP #AREA,#0,#SHR1 ;Try to open a file which is access locked
        BCC     1$             ;Branch if OK
        .PRINT  #LKPERR        ;Say couldn't get the file
        BR      3$             ;Go on to try second file

1$:      MOV     #SHRFIL,RO      ;Point to EMT arg block to
        EMT     375             ;Declare file for shared access
        BCC     2$             ;If got the file, branch to read it
        CALL    EXPLER         ;Else explain why
        .PRINT  #INSHR1        ;Say we can't share SHR1
        .CLOSE  #0             ;Release channel 0
        BR      3$             ;Go on to next file

2$:      .READW  #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Try to read block 0 of SHR1
        .PRINT  #BUFFER        ;Display it for kicks
        .CLOSE  #0             ;Done with SHR1 for the moment

3$:      .LOOKUP #AREA,#0,#SHR2 ;Try to open SHR2
        BCC     4$             ;Branch if OK
        .PRINT  #LKPER2        ;Say we couldn't even open it
        BR      6$             ;Go on to try SHR1 again

4$:      MOV     #SHRFIL,RO      ;Point to EMT arg block to
        EMT     375             ;Declare file for shared access
                                   ;Access Input, Update to show lockout
                                   ; though we don't write in this example
        BCC     5$             ;Branch if we can share it
        CALL    EXPLER         ;Explain why not
        .PRINT  #AGAIN         ;Say we will try again later
        .TWAIT  #AREA,#TIME    ;Wait 5 seconds and
        BR      4$             ;Try again

5$:      .READW  #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Read block 0 of SHR2
        .PRINT  #BUFFER        ;Prove that we got it
        .CLOSE  #0             ;Done with SHR2
```


Shared File Record Locking

```

6$:    .LOOKUP #AREA,#0,#SHR1 ;Try SHR1 again
      BCC      7$              ;Branch if it worked
      .PRINT   #LKPERR         ;Say we couldn't do it
      .EXIT

7$:    MOV      #SHRFIL,R0      ;Point to EMT arg block to
      EMT      375              ;Declare shared file
      BCC      10$             ;Branch if OK
      CALL     EXPLER          ;And explain the error
      .PRINT   #STLLOK        ;Say it was still locked
      BR       11$            ;And quit

10$:   .READW   #AREA,#0,#BUFFER,#BUFSIZ,#1 ;Read in block 1
      .PRINT   #BUFFER         ;And display it
11$:   .CLOSE   #0              ;Done with SHR1

      .EXIT

EXPLER: MOVB    @#ERRBYT,R0     ;Find out why can't share it
      DEC      R0              ;Convert to zero index
      ASL      R0              ;Make into word offset
      .PRINT   SHRERR(R0)      ;Say why we couldn't get it
      RETURN

      .NLIST   BEX

AREA:   .BLKW   10              ;EMT arg block
SHRFIL: .BYTE   0,125           ;EMT arg block to declare file shared
      .WORD    5               ;Access Shared, Update
SHRERR: .WORD    NOTOPN         ;Shared file error message pointers
      .WORD    XSSFCH
      .WORD    XSSFOP
      .WORD    AXSCON

TIME:   .WORD    0,5.*60.       ;5.sec * 60.tics/sec
SHR1:   .RAD50   /DK SHR1  DAT/ ;Input file #1 name
SHR2:   .RAD50   /DK SHR2  DAT/ ;Input file #2 name
NOTOPN: .ASCIZ   /Cannot share unopened file/
XSSFCH: .ASCIZ   /Too many channels opened to shared files/
XSSFOP: .ASCIZ   /Too many shared files open/
AXSCON: .ASCIZ   /Shared file protected by another job/
INSHR1: .ASCIZ   /On first try at SHR1.DAT/
LKPERR: .ASCIZ   /Unable to lookup SHR1.DAT/
LKPER2: .ASCIZ   /Unable to lookup SHR2.DAT/
AGAIN:  .ASCIZ   /Can't access SHR2.DAT, will try again in 5 seconds/
STLLOK: .ASCIZ   /On second try at SHR1.DAT/
      .EVEN

BUFFER: .BLKW    BUFSIZ         ;Input read buffer

      .END    START

```


Shared File Record Locking

See also the example program SHARED in the section on opening a shared file.

9.3 Waiting for a locked block

The following EMT can be used to lock a specific block in a file. If the requested block is locked by another job, the requesting job will be suspended until the desired block becomes available. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following area:

```
.BYTE    chan,102
.WORD     block
```

where "chan" is the number of an I/O channel that has previously been declared to be open to a shared file and "block" is the number of the block in the file to be locked. Other blocks in the file which were previously locked remain locked. The maximum number of blocks which may be simultaneously held locked is specified when TSX-Plus is generated. A block number of -1 (octal 177777) can be used to request that all blocks in the file be locked. If several users request the same block, access will be granted sequentially in the order that the requests are received.

Errors:

<u>Code</u>	<u>Meaning</u>
1	Channel is not open to a shared file
2	Request to lock too many blocks in file

Example:

```
.TITLE  LOCKW
.ENABL  LC
```

; This program cooperates with the example program (LOCK) in the next
; section to demonstrate shared file record locking.

```
.MCALL .PRINT,.EXIT,.TWAIT,.LOOKUP,.READW,.CLOSE
```

```
ERRBYT = 52          ;EMT error code byte
BUFSIZ = 256.         ;Words per disk block
```

```
START: .LOOKUP #AREA,#0,#SHR1 ;Open SHR1.DAT
       BCC     l$             ;Branch if OK
       .PRINT  #LKPERR        ;Say bad lookup
       .EXIT
```

```
l$:    MOV     #SHRFIL,R0      ;Point to EMT arg block to
       EMT     375             ;Declare shared file
```


Shared File Record Locking

```

BCC      3$      ;Branch if OK
MOVB     @#ERRBYT,R0 ;Get the error code
DEC      R0      ;Make zero index
ASL      R0      ;Convert to word offset
.PRINT   SHRERR(R0) ;Print the error message
2$:      .CLOSE  #0      ;Give back the channel
        .EXIT

3$:      MOV     #LOCKW,R0 ;Point to EMT arg block to
        EMT     375      ;Lock block 0 of SHR1.DAT
        ;(Job is suspended until block available to be locked)
BCC      5$      ;Branch when block is ready
CMPB     @#ERRBYT,#1 ;Which error?
BHI      4$      ;Too many blocks locked?
.PRINT   #SFCNOP    ;Wasn't open to shared file!
BR       2$      ;Give up
4$:      .PRINT  #XSLKBL ;Too many locked blocks in file
        ;(Defined by MXLBLK parameter in TSGEN)
BR       2$      ;Give up

5$:      .PRINT  #PROMPT ;Switch lines to attempt access
        .TWAIT  #AREA,#SEC20 ;Wait 20 seconds before unlocking
MOV      #UNLOCK,R0 ;Point to EMT arg block to
EMT      375      ;Unlock a single block
BCC      6$      ;Branch if OK
.PRINT   #SFCNOP    ;Wasn't shared file!
BR       2$      ;Give up

6$:      .TWAIT  #AREA,#SEC10 ;Wait 10 seconds for companion
MOV      #CKWSHR,R0 ;Point to EMT arg block to
EMT      375      ;Check for writes to shared file
BCC      8$      ;If none, wrap up
.PRINT   #CHANGD    ;Say we have new data
.READW   #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Get block 0
.PRINT   #BUFFER    ;Show current contents block 0
.READW   #AREA,#0,#BUFFER,#BUFSIZ,#1 ;Get block 1
.PRINT   #BUFFER    ;Show contents block 1
BR       DONE

8$:      .PRINT  #NOCHNG ;Say nothing has changed

DONE:    .CLOSE  #0      ;Free up channel
        .EXIT

.NLIST   BEX
AREA:    .BLKW   10      ;EMT arg block area
SHRFIL:  .BYTE   0,125   ;EMT arg block to declare shared file
        .WORD   4       ;Access Shared, Input
LOCKW:   .BYTE   0,102   ;EMT arg block to lock shared file block
        .WORD   0       ;Block number to be locked

```


Shared File Record Locking

```
UNLOCK: .BYTE 0,113 ;EMT arg block to unlock shared file block
        .WORD 0 ;Block number to be unlocked
CKWSHR: .BYTE 0,121 ;EMT arg block to check writes to shared file
SHRERR: .WORD NOTOPN ;Shared file error message table
        .WORD XSSFCH
        .WORD XSSFOP
        .WORD AXSCON
SHR1: .RAD50 /DK SHR1 DAT/ ;File name to be shared
SEC20: .WORD 0,20.*60. ;20.sec * 60.tics/sec
SEC10: .WORD 0,10.*60. ;10.sec * 60.tics/sec
NOTOPN: .ASCIZ /Channel not opened to shared file/<7>
XSSFCH: .ASCIZ /Too many channels opened to shared files/<7>
XSSFOP: .ASCIZ /Too many shared files open/<7>
AXSCON: .ASCIZ /File protection access conflict/<7>
LKPERR: .ASCIZ /Unable to open SHR1.DAT/
SFCNOP: .ASCIZ /Can't lock or unlock block not open to shared file/
XSLKBL: .ASCIZ /Can't lock so many blocks in one file/
CHANGED: .ASCIZ /Data has been written to file. Contents follow:/
NOCHNG: .ASCIZ /Data in file is unchanged/
PROMPT: .ASCII /Block 0 in SHR1.DAT will remain locked for 20 sec/<15><12>
        .ASCIZ /Go to another line and RUN TLOCK to test it/<7>
        .EVEN
BUFFER: .BLKW BUFSIZ

        .END START
```

9.4 Trying to lock a block

This EMT is similar in operation to the previous EMT: it too is used to request that file blocks be locked. The difference is that if the requested block is already locked by another user the previous EMT suspends the requesting program whereas this EMT does not suspend the program but rather returns an error code. As above, a request to lock block #-1 is treated as a request to lock the entire file. If the block is available it is locked for the requesting user and no error is reported. The form of this EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE chan,103
.WORD block
```

where "chan" is the number of the I/O channel associated with the file and "block" is the number of the block which is to be locked.

Errors:

Code	Meaning
----	-----
1	Channel is not open to a shared file.
2	Request to lock too many blocks in file.
3	Requested block is locked by another user.

Example:

```
.TITLE LOCK
.ENABL LC
```

```
; This program cooperates with the example program (LOCKW) in the
; previous section to demonstrate shared file record locking.
```

```
.MCALL .PRINT,.EXIT,.WRITW,.TWAIT,.LOOKUP,.CLOSE
```

```
ERRBYT = 52                                ;Error byte address

START: .LOOKUP #AREA,#0,#SHR1 ;Try to open SHR1.DAT
      BCC 1$ ;Branch if OK
      .PRINT #LKPERR ;Say we couldn't open
      .EXIT

1$: MOV #SHRFIL,R0 ;Point to EMT arg block to
   EMT 375 ;Declare shared file
   BCC 3$ ;Branch if OK
   MOVB @#ERRBYT,R0 ;Get error type
   DEC R0 ;Convert to zero index
   ASL R0 ;Make into word offset
   .PRINT SHRERR(R0) ;Display the error type
2$: .CLOSE #0 ;Release the channel
   .EXIT ;And give up

3$: MOV #LOCK,R0 ;Point to EMT arg block to
   EMT 375 ;Try to unlock block 0
   BCC 6$ ;Branch if OK
   CMPB @#ERRBYT,#2 ;Which error was it
   BLO 4$ ;Wasn't open to share file?
   BEQ 5$ ;Request to open too many blocks in file?
   .PRINT #WAITNG ;Block locked by another user
   .TWAIT #AREA,#TIME ;Wait 3 seconds
   BR 3$ ;And try again
4$: .PRINT #NOPNSF ;Not open to shared file
   BR 2$ ;Give up
5$: .PRINT #XSLKBL ;Too many blocks locked in file
   BR 2$ ;Give up

6$: .WRITW #AREA,#0,#BUFFER,#BUFSIZ,#0 ;Rewrite block 0
   MOV #UNLALL,R0 ;Point to EMT arg block to
```


Shared File Record Locking

```

      EMT      375                ;Release all blocks locked by this program
      .PRINT   #GOBACK           ;Message: done, go back to original line
      BR       2$                ;Done

      .NLIST   BEX

AREA:  .BLKW   10                ;EMT arg block
SHR1:  .RAD50  /DK SHR1  DAT/    ;Name of shared file
SHRFIL: .BYTE  0,125             ;EMT arg block to share file on chan 0
      .WORD    3                ;Access Protected, Update
LOCK:  .BYTE   0,103            ;EMT arg block to lock block on chan 0
      .WORD    0                ;number of block to be locked
UNLALL: .BYTE   0,101            ;EMT arg block to unlock all blocks on chan 0
                                      ;(Only applies to blocks locked by this job)
TIME:  .WORD   0,3.*60.         ;3.sec * 60.tics/sec
SHRERR: .WORD   SFCNOP           ;File sharing EMT error table
      .WORD    XSSFCN
      .WORD    XSSFOP
      .WORD    AXSCON

SFCNOP: .ASCIZ  /Channel not open to file/<7>
XSSFCN: .ASCIZ  /Too many channels open to shared files/<7>
XSSFOP: .ASCIZ  /Too many shared files open/<7>
AXSCON: .ASCIZ  /Shared file access conflict/<7>
LKPERR: .ASCIZ  /Couldn't open SHR1.DAT/<7>
WAITNG: .ASCII  /Requested block not available for locking/<15><12>
      .ASCIZ    /Will try again in 3 seconds/
GOBACK: .ASCIZ  /Done, log off and go back to original line/
NOPNSF: .ASCIZ  /Channel not open to shared file/<7>
XSLKBL: .ASCIZ  /Attempt to lock too many blocks in file/<7>
      .EVEN
BUFFER: .ASCII  /(SHR1)This line was written by the program LOCK./
      .ASCIZ    /                               /<15><12>
BUFSIZ = <.-BUFFER+1>/2        ;Number of words to write
      .BYTE     0,0             ;Safety bumper

      .END      START

```

9.5 Unlocking a specific block

The following EMT is used to unlock a specific block in a file. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```

      .BYTE    chan,113
      .WORD    block-number

```

where "chan" is the number of the I/O channel opened to the shared file and "block-number" is the number of the block to be unlocked.

Errors:

<u>Code</u>	<u>Meaning</u>
1	Specified channel not opened to a shared file

Example:

See the example program LOCKW in the section on waiting for a locked block.

9.6 Unlocking all locked blocks in a file

The following EMT is used to unlock all blocks held locked in a file. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

.BYTE chan,101

where "chan" is the I/O channel number open to the shared file. When this EMT is executed all blocks previously locked by the user on the shared file are unlocked. Blocks locked by the user on other files are not released nor are blocks of the same file that are locked by other users.

Errors:

<u>Code</u>	<u>Meaning</u>
1	Channel is not open to a shared file.

Example:

See the example program LOCK in the section on trying to lock a block.

9.7 Checking for writes to a shared file

The following EMT can be used to determine if any other user has written to a shared file. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE chan,121

where "chan" is the I/O channel number opened to the shared file. If no other user has written to the file since the file was opened by the user issuing this EMT or since that last time this EMT was issued for the file, the carry-flag is clear on return from the EMT.

Shared File Record Locking

Errors:

<u>Code</u>	<u>Meaning</u>
2	Some other job has written to file since last check

This EMT is useful when data from a shared file is being held in a program buffer. If no other user has written to the file, then the data is still valid. However, if the data in the file has been re-written then it must be re-read. The usual sequence of operations in this situation is to first lock the block whose data is in the program's buffer, then do the EMT to see if the file has been written to. If the file has not been modified the data in memory is valid and can be used, otherwise the block must be re-read from the file.

Example:

See the example program LOCKW in the section on waiting for a locked block.

9.8 Data caching

Data caching is a technique provided by TSX-Plus to speed access to files. When TSX-Plus is generated a certain number of 512-byte buffer areas may be set aside for data caching. These buffer areas are part of the resident system data area and are not associated with any particular job. There are two kinds of data caching: generalized data caching; and shared-file data caching. Both kinds may be used automatically with minimal intervention on the part of the programmer or operator. Generalized data caching applies to all files on MOUNTed devices, while shared-file data caching applies only to files which have been declared as shared files. Generally, only one of these types is selected during generation of a TSX-Plus system. The following discussion applies to shared-file data caching. See the TSX-Plus System Manager's Guide for more information on data caching.

Each time a request is issued to read a shared file, a check is made to see if the blocks being read are currently stored in the data cache. If so, the data is moved from the cache buffer to the program buffer and no disk I/O operations are performed. When data in the cache buffers is accessed, a use count is incremented. Periodically, the use counts for all buffers is divided by two. If the data blocks being read are not currently in the cache, the data is read from the disk into the program buffer and then it is moved into the cache buffers with the lowest use count.

When a write operation is done to a file that is being cached, a check is made to see if the data being written is currently stored in the cache. If so, the cache buffers are updated. In any case the data is written to the disk. In other words, this is a "write-through" cache; the disk file is always updated and caching does not improve the performance of "writes".

All data files that are declared to TSX-Plus for shared access (using EMT 375 with function code 125) are eligible for data block caching regardless of their access protection type. Data caching on a shared file may be disabled by

Shared File Record Locking

setting bit 8 (octal 400) in the access-code word of the EMT argument block when the file is declared for shared access. Data caching is particularly effective for COBOL-Plus ISAM files.

1940-1941
1942-1943
1944-1945
1946-1947
1948-1949
1950-1951
1952-1953
1954-1955
1956-1957
1958-1959
1960-1961
1962-1963
1964-1965
1966-1967
1968-1969
1970-1971
1972-1973
1974-1975
1976-1977
1978-1979
1980-1981
1982-1983
1984-1985
1986-1987
1988-1989
1990-1991
1992-1993
1994-1995
1996-1997
1998-1999
2000-2001
2002-2003
2004-2005
2006-2007
2008-2009
2010-2011
2012-2013
2014-2015
2016-2017
2018-2019
2020-2021
2022-2023
2024-2025

10. MESSAGE COMMUNICATIONS FACILITIES

TSX-Plus provides an optional facility that allows running programs to send messages to each other. This message communication facility allows programs to send messages through named channels, check to see if messages are pending, and suspend execution until a message is received. TSX-Plus provides EMTs for each of these operations which are described below.

10.1 Message channels

Messages are transferred to and from programs by using TSX-Plus "Message Channels". A message channel accepts a message from a sending program, stores the message in a queue associated with the channel and delivers the message to a receiving program when requested. Message channels are totally separate from I/O channels.

Each message channel is identified to the sending and receiving programs by a one to six character name. The total number of message channels is defined when TSX-Plus is generated. The names associated with the channels are defined dynamically by the running programs. A message channel is said to be "active" if any messages are being held in the queue associated with the channel or if any program is waiting for a message from the channel. When message channels become inactive they are released and may be reused.

Once a message is queued on a channel, that message will remain in the queue until some program receives it or the TSX-Plus system is halted. A program may exit after queuing a message without affecting the queued message. This allows one program to leave a message for another program that will run later.

10.2 Sending a message

The following EMT is used to queue a message on a named channel. If other messages are already pending on the channel, the new message is added to the end of the list of waiting messages. The sending program continues execution after the EMT and does not wait for the message to be accepted by a receiving program. During processing of the EMT the message is copied to an internal buffer, and the sending program is free to destroy its message on return from the EMT. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE    0,104  
.WORD    chadr  
.WORD    msadr  
.WORD    mssiz
```

where "chadr" is the address of a six byte field containing the name of the message channel (ASCII with trailing blanks if the name is less than six characters), "msadr" is the address of the beginning of the message text, and "mssiz" is the message length in bytes.

Message Channels

Errors:

Code	Meaning
1	All message channels are busy.
2	Maximum allowed number of messages already in message queues.
4	The transmitted message is too long. The message is truncated to maximum allowed length.
5	Maximum number of message requests pending.

The system manager may alter parameters during TSX-Plus generation to alleviate these error conditions.

Example:

```

        .TITLE SNDMSG
        .ENABL LC

; Demonstrates use of the TSX-Plus EMT to queue a message to the interprocess
; message communication facility.

        .MCALL .EXIT,.PRINT,.GTLIN

ERRBYT = 52                                ;EMT error byte

START:  .GTLIN  #MSGBUF,#MSGPRT ;Get the message to be queued
        MOV     #MSGBUF,R1      ;Point to beginning of buffer
1$:      TSTB    (R1)+           ;Find end of message
        BNE     1$              ;
        SUB     #<MSGBUF+1>,R1  ;Determine message length
                                ;accounting for post-increment
        MOV     R1,MSGLEN       ;Set message length in EMT arg block
        .GTLIN  #CNLBUF,#CNLPRT ;Get the six character channel name
        MOV     #MSGBLK,R0      ;Point to EMT arg block to
        EMT     375             ;Send message on named channel
        BCC     9$              ;Branch if no error
        MOVB    @#ERRBYT,R0     ;Which error?
        DEC     R0              ;Zero offset
        ASL     R0              ;Convert to word index
        .PRINT  ERR_TBL(R0)     ;Display the appropriate message
        .EXIT

9$:      CLRB     CNLBUF+6       ;Make channel name ASCII
        .PRINT   #DONEOK        ;Inform user message queued
        .PRINT   #CNLBUF        ;on channel CNLBUF
        .EXIT

        .NLIST  BEX
MSGBLK:  .BYTE   0,104           ;EMT block: send message on named channel
        .WORD    CNLBUF         ;Address of channel name
        .WORD    MSGBUF         ;Address of message

```



```

MSGLEN: .WORD    0                ;Char length of message
ERRTBL: .WORD    BSYERR           ;Table of send error messages
        .WORD    FULERR
        .WORD    OHOH             ;Error code 3 not used
        .WORD    TRNERR
BSYERR: .ASCIZ    /?SNDMSG-F-Maximum number of messages have been queued./
FULERR: .ASCIZ    /?SNDMSG-F-All message channels are busy./
OHOH:   .ASCIZ    /?SNDMSG-F-This is a non-existent error./
TRNERR: .ASCIZ    /?SNDMSG-W-Message was too long, truncated./
MSGPRT: .ASCIZ    /Message to be queued: /
CNLPRT: .ASCII    <15><12>/Channel Name (six characters max): /<200>
DONEOK: .ASCII    /Message queued on channel /<200>
        .EVEN
CNLBUF: .BLKB     80.             ;First 6 chars to contain file name
MSGBUF: .BLKB     80.             ;Message buffer.

        .END      START

```

10.3 Checking for pending messages

The following EMT is used to receive a message from a named channel if a message is pending on the channel. If no message is pending, an error code (3) is returned, and the program is allowed to continue execution. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE    0,105
.WORD    chadr
.WORD    msadr
.WORD    mssiz

```

where "chadr" points to a field with a six character channel name, "msadr" points to the buffer in which the message is to be placed, and "mssiz" is the size of the message buffer (bytes).

If a message is received, its length (bytes) is placed in R0 on return from the EMT. If the message is longer than the message buffer (mssiz), only the first part of the message will be received.

Errors:

Code	Meaning
----	-----
1	All message channels are busy.
3	No message was queued on the named channel.
4	Message was longer than the receiving buffer.
5	Maximum number of message requests pending.

Message Channels

Example:

```
.TITLE GETMSG

; Demonstrates use of the TSX-Plus EMT to check for pending messages in the
; interprocess message communication facility.

.MCALL .EXIT,.PRINT,.GTLIN

ERRBYT = 52                                ;EMT error byte

START: .GTLIN #CNLBUF,#CNLPRT              ;Get the channel name
      MOV     #MSGBLK,R0                   ;Put EMT argument block address in R0
      EMT     375                          ;EMT to check channel for message
      BCC     5$                          ;Error?
      CMPB    @#ERRBYT,#4                 ;Only two errors possible
      BEQ     2$                          ;Overflow message buffer?
      .PRINT  #NOMERR                    ;No message
      .EXIT

2$:   .PRINT  #TRNERR                    ;Print truncation warning
5$:   .PRINT  #PNDMSG                    ;Print message preamble
      .PRINT  #MSGBUF                    ;Print actual message
      .EXIT

MSGBLK: .BYTE  0,105                      ;GETMSG EMT block
      .WORD   CNLBUF                      ;Channel name buffer address
      .WORD   MSGBUF                      ;Buffer address to receive message
      .WORD   81.                        ;Buffer length
      .NLIST  BEX

CNLBUF: .BLKB  80.                        ;First 6 chars are channel name
MSGBUF: .BLKB  80.                        ;Message buffer
      .WORD   0                          ;Insure ASCIIZ

CNLPRT: .ASCII /Channel Name (6 chars): /<200>
NOMERR: .ASCIIZ /?GETMSG-F-No messages pending in named channel./
TRNERR: .ASCIIZ /?GETMSG-W-Message truncated/<7>
PNDMSG: .ASCIIZ /Message pending in named queue is:/

.END    START
```

10.4 Waiting for a message

The following EMT is used to suspend execution of a program until a message becomes available on a named channel. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE 0,106
.WORD chadr
.WORD msadr
.WORD mssiz
```

where "chadr" points to a six byte field containing the channel name, "msadr" points to the buffer where the message is to be placed, and "mssiz" is the size of the message buffer (bytes).

The length of the received message (bytes) is placed in R0 on return from the EMT.

Errors:

Code	Meaning
-----	-----
1	All message channels are busy.
4	Message was longer than the receiving buffer.
5	Maximum number of message requests pending.

Example:

```
.TITLE WATMSG
.ENABL LC
```

; Demonstrate TSX-Plus EMT to wait for a queued message from the
; interprocess message communication facility.

```
.MCALL .EXIT,.PRINT,.GTLIN
```

```
ERRBYT = 52 ;EMT error byte

START: .GTLIN #CNLBUF,#CNLPRT ;Get the channel name
       .PRINT #WAITNG ;Explain waiting
       MOV #MSGBLK,R0 ;Put EMT argument block address in R0
       EMT 375 ;EMT to check channel for message
       BCC 5$ ;Check for error
       CMPB @#ERRBYT,#1 ;Error?
       BHI 2$ ;Message truncated
       .PRINT #NOMERR ;All channels busy
       .EXIT

2$: .PRINT #TRNERR ;Print truncation warning
5$: .PRINT #RCVMSG ;Print message preamble
    .PRINT #MSGBUF ;Print actual message
    .EXIT

MSGBLK: .BYTE 0,106 ;WATMSG EMT block
```


Message Channels

```
.WORD CNLBUF ;Channel name buffer address
.WORD MSGBUF ;Buffer address to receive message
.WORD 81. ;Buffer length
CNLBUF: .BLKB 80. ;Channel name first 6 chars
MSGBUF: .BLKB 80. ;Message buffer
.WORD 0 ;Insure ASCIIZ
.NLIST BEX
CNLPRT: .ASCII /Channel Name (6 chars): /<200>
WAITNG: .ASCII /Waiting for a message . . ./<15><12>
.ASCIZ /Go to another line and send me something./
NOMERR: .ASCIZ /?WATMSG-F-All message channels are busy./
TRNERR: .ASCIZ /?WATMSG-W-Message truncated./<7>
RCVMSG: .ASCIZ /Message received in named queue is:/

.END START
```

10.5 Scheduling a message completion routine

The following EMT is used to schedule a completion routine to be entered when a message is received on a named channel. The form of the EMT is:

```
EMT 375
```

with R0 pointing to an argument block of the form:

```
.BYTE 1,106
.WORD chadr
.WORD msadr
.WORD mssiz
.WORD cplrtn
```

where "chadr" points to a six byte field containing the message channel name, "msadr" is the address of the buffer where the message is to be placed, "mssiz" is the size of the message buffer (bytes), and "cplrtn" is the address of the completion routine to be entered when a message is received.

On entry to the completion routine, R0 contains the number of bytes in the message that was received and R1 contains the number of the job that sent the message. Another message completion routine may be scheduled from within the completion routine if desired.

The .SPND and .RSUM system service calls may be used with this facility if the program reaches a point where it must suspend its execution until a message arrives.

Errors:

<u>Code</u>	<u>Meaning</u>
1	All message channels are busy
4	Message was longer than the receiving buffer.
5	Maximum number of message requests pending.

Example:

```

        .TITLE  MSGCPL
        .ENABL  LC
;
; Demonstrate message completion routine
; Do processing while waiting for a message,
; then display the message and repeat.
;
        .MCALL  .PRINT,.EXIT,.TTYOUT
        .GLOBL  PRTDEC
        .DSABL  GBL
;
ERRBYT  = 52                ;EMT error byte address
SPACE   = 40                ;ASCII space char
BELL    = 7                 ;ASCII bell char
REPTS   = 50                ;Outer idle loop repeat counter
WIDTH   = 32.               ;Screen formatting width
;
; Schedule message completion routine
;
START:  MOV     #MSGCPL,R0    ;Point to EMT arg block to
        EMT     375          ;Schedule completion routine
        BCC     2$           ;Branch if OK
        MOVB    @#ERRBYT,R1  ;Get error code
1$:     ASL      R1           ;Convert to word index
        .PRINT  SCHERR(R1)    ;Print appropriate error message
        .EXIT                ;Quit
;
; Idle processing loop while waiting for completion routine
;
2$:     MOV      #REPTS,R2    ;Outer idle loop counter
3$:     MOV      #-1,R3       ;Inner idle loop counter
        TST      MESSAG      ;Any message pending?
        BEQ      5$          ;Do other processing if not
        BMI      4$          ;Msg compl rtn error?
        CALL     DSPMSG      ;No error, display message
        BR       2$          ;Continue waiting for next
4$:     MOV      MESSAG,R1    ;Recover error code
        BR       1$          ;Go print error and die
5$:     SOB      R3,5$        ;Inner idle loop
        SOB      R2,3$        ;Outer idle loop
        .TTYOUT  #BELL        ;Indicate activity

```


Message Channels

```

        BR      2$                ;And repeat forever
;
;  Display the received message
;
DSPMSG: MOV     R3,-(SP)
        .PRINT  #CRLF             ;Format display
        MOV     MSGLEN,R2         ;Get length count
        MOV     R2,R0             ;Again for display
        CALL    PRTDEC            ;Show message size
        .PRINT  #MSGFRM           ;"byte message rec'd from job #"
        MOV     JOBNUM,R0         ;Get sending job number
        CALL    PRTDEC            ;Display job number
        MOV     #MSADR,R1         ;Get pointer to message
        ADD     R1,R2             ;Save pointer to message end
1$:     MOV     #WIDTH,R4         ;Display-width counter
        .PRINT  #LEFT             ;Format display
2$:     MOVB    (R1)+,R0           ;Get next character
        CMPB    R0,#SPACE         ;Printable?
        BGE     3$               ;Branch if so
        .TTYOUT #'^'             ;Mark control character
        DEC     R4               ;Sub 1 from width
        MOVB    -1(R1),R0         ;Recover character
        ADD     #'@,R0           ;Convert to uppercase char
3$:     .TTYOUT             ;Display char
        CMP     R1,R2             ;End of display?
        BHIS    4$               ;Branch if so
        SOB     R4,2$            ;Continue unless need to wrap
        .PRINT  #RIGHT            ;Format display
        BR      1$               ;Next line
4$:     .PRINT  #RIGHT            ;Format display
        CLR     MESSAG           ;Say we are done with this one
        MOV     (SP)+,R3
        RETURN
;
;  Completion routine to be entered when message received
;
CPLRTN: TST     MESSAG           ;Done with last message?
        BNE     1$               ;Branch (and lose this one) if not
        MOV     R0,MSGLEN         ;Save message length
        MOV     R1,JOBNUM         ;Save sending job number
        MOV     #1,MESSAG         ;Flag message received
1$:     MOV     #MSGCPL,R0        ;Point to EMT arg block to
        EMT     375               ;Reschedule myself
        BCC     2$               ;Branch if OK
        MOVB    @#ERRBYT,MESSAG   ;Else save error code
        BIS     #100000,MESSAG    ;And flag it as error
2$:     RETURN
;
;  EMT arg blocks and word buffers
;

```


Message Channels

```

MSGCPL: .BYTE    1,106          ;EMT arg block - msg compl rtn
        .WORD    CHANAM        ;Pointer to channel name
        .WORD    MSADR         ;Address of message buffer
        .WORD    MSEND-MSADR   ;Size of message buffer
        .WORD    CPLRTN        ;Address of completion routine
JOBNUM:  .WORD    0             ;Cell for # of message sender
MSGLEN:  .WORD    0             ;Cell for received message length
MESSAG:  .WORD    0             ;Cell for msg rcvd signal
SCHERR:  .WORD    0             ;Msg compl rtn sched error table
        .WORD    NOFREE        ; 1 No free message channels
        .WORD    UNUSED        ; 2 Max messages already queued
        .WORD    UNUSED        ; 3 No message queued on channel
        .WORD    TOOLNG        ; 4 Message overflowed buffer
        .WORD    NOMRB         ; 5 Max messages requests pending

```

```

;
; Messages and byte buffers
;

```

```

        .NLIST  BEX
CHANAM: .ASCII  /CHANEL/        ;Name of watched message channel
NOFREE: .ASCIZ  /All message channels are busy./
UNUSED: .ASCIZ  /MSGCPL should never generate this error./
TOOLNG: .ASCIZ  /Message was longer than message buffer./
NOMRB:  .ASCIZ  /Maximum number of message requests pending./
MSGFRM: .ASCII  / byte message received from job number /<200>
CRLF:   .BYTE   15,12,200
LEFT:   .ASCII  <15><12>/--/<76><200>
RIGHT:  .ASCII  <74>/--/<200>
MSADR:  .BLKB   200.            ;Message buffer
MSEND:

        .LIST   BEX
        .EVEN
        .END    START

```


1. The first part of the report is a summary of the work done during the year. It includes a list of the projects completed and a brief description of the results. The second part is a detailed account of the work done on each project. It includes a description of the objectives, the methods used, and the results obtained. The third part is a discussion of the results and a comparison with the results of other workers. The fourth part is a list of the references.

Summary of the work done during the year

The work done during the year was divided into four main parts. The first part was a summary of the work done during the year. The second part was a detailed account of the work done on each project. The third part was a discussion of the results and a comparison with the results of other workers. The fourth part was a list of the references.

11. REAL-TIME PROGRAM SUPPORT

TSX-Plus provides a real-time program support facility that allows multiple real-time programs to run concurrently with normal time-sharing operations. The basic functions provided by this facility are summarized below.

1. The ability to map the I/O page into the user's virtual memory region so that device status and control registers may be directly accessed by the program.
2. The ability to connect device interrupt vectors to program interrupt service routines running at fork level or to program completion routines running at user-selectable priority levels with full job context.
3. The ability for a program to lock itself in memory so that rapid interrupt response can be assured.
4. The ability for a program to suspend its execution until an interrupt occurs.
5. The ability to set execution priorities for tasks.
6. The ability to convert a virtual address within the job's region to a physical address for DMA I/O control.
7. The ability to map a virtual address region to a physical address region.
8. The ability for a program to declare a list of addresses of device control registers to be reset when the program exits or aborts (.DEVICE EMT).

Real-time support features are only available if the real-time support facility is included in TSX-Plus when the system is generated.

A program must have operator privilege to use any of the real-time features described in this chapter. The real-time facilities are available to both normal jobs controlled by time-sharing lines and to detached jobs. Note that detached jobs that are specified during system generation for automatic startup run with operator privilege; detached jobs started by time-sharing users have operator privilege only if the user starting them does.

Each of the EMTs described in this chapter will return with the carry bit set and an error code of 0 if real-time support was not included during system generation or if the job is not privileged.

Real-Time Programs

11.1 Accessing the I/O page

A basic facility required by most real-time programs is the ability to access the PDP-11 I/O page (160000-177777) which contains the device control and status registers. Under TSX-Plus, addresses in this range are normally mapped to a simulated RMON or may be used as normal program space. This is done since many old programs require direct access to certain system values at fixed offsets into RMON, although recent programs should access these values with the .GVAL and .PVAL EMTs. TSX-Plus provides several EMTs to deal with mapping of the I/O page and accessing locations within it. These are discussed below. See Chapter 8 for a discussion of various mapping techniques which may be used under TSX-Plus. The /IOPAGE switch to the R[UN] command may also be used to map a program's PAR 7 to the I/O page.

A TSX-Plus real-time program can access the I/O page in one of two ways: It can cause the program's virtual address region in the range 160000 to 177777 to be mapped directly to the I/O page so that it can directly access device registers; or it can leave the virtual address range mapped to the simulated RMON and use a set of EMTs to peek, poke, bit-set and bit-clear registers in the I/O page. It is much more efficient to directly access the device control registers by mapping the I/O page into the program's virtual address region than to use EMTs to perform each access. However, this technique will not work if the program must also directly access offsets inside RMON. The correct way for a program to access RMON offsets is to use the .GVAL EMT which will work even if the I/O page is mapped into the program region.

11.1.1 EMT to map the I/O page into the program space.

The following EMT can be used to cause the program's virtual address region in the range 160000 to 177777 to be mapped to the I/O page. The /IOPAGE switch to the R[UN] command may also be used to map a program's PAR 7 to the I/O page. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    5,140
```

The I/O page mapping set up by this EMT remains in effect until the program exits, chains, or the EMT described in the next section is used to remap to RMON. Note that completion routines and interrupt service routines run with the same memory mapping as the main-line code of the job.

The .GVAL EMT with offset value -8. may be used to determine if PAR 7 is currently mapped to the I/O page or to the simulated RMON. See the description in Chapter 7 of the special use of .GVAL for further information.

Example:

```
.TITLE  MAPIOP
.ENABL  LC
```

```
;Demonstrate TSX-Plus EMTs to map to the I/O page and back to RMON
```

```
RMONST = 54          ;SYSCOM location holding base of RMON
CONFIG = 300         ;Fixed offset into RMON of CONFIG word
RCSR    = 176540     ;Serial line RCSR address
RBUF    = RCSR+2     ;Line input buffer address
CTRLC   = 3          ;Control-C

.MCALL  .PRINT,.EXIT,.TTYOUT,.GVAL
.GLOBL  PRTOCT

START:  CALL    SHOMAP      ;Display current PAR 7 mapping
        CALL    SHODAT     ;Demonstrate "RMON" mapping

        .PRINT  #IOPMSG    ;Say we are switching to I/O page mapping
        MOV     #MAPIOP,R0 ;Point to EMT arg block to
        EMT     375        ;Map to I/O page
        BCC     1$         ;Branch if OK to map
        .PRINT  #NOPRIV    ;You aren't allowed to do that
        .EXIT

1$:     CALL    SHOMAP      ;Display current PAR 7 mapping
        .PRINT  #TYPE      ;Prompt for input from I/O page
        MOV     @#RCSR,CSRSV ;Save a copy of current CSR
        CLR     @#RCSR     ;Disable interrupts on serial line
2$:     TSTB    @#RCSR     ;Is anything available from the line
        BPL     2$         ;No, keep checking
        MOVB    @#RBUF,R0  ;Get the new character
        CMPB    R0,#CTRLC  ;Should we quit?
        BEQ     3$         ;Quit on ^C
        .TTYOUT      ;Display the character
        BR      2$         ;And repeat

3$:     MOV     CSRSV,@#RCSR ;Restore original CSR
        .PRINT  #GOBACK    ;Say we are returning to original mapping
        MOV     #MAPMON,R0 ;Point to EMT arg block to
        EMT     375        ;Map back to simulated RMON
        CALL    SHOMAP     ;Display current PAR 7 mapping
        CALL    SHODAT     ;And prove we are back
        .EXIT

SHOMAP: .PRINT  #MAPMSG    ;Current mapping preface
        .GVAL   #AREA,#-8. ;What is our current PAR 7 mapping?
        ASL     R0         ;Convert to word offset
        .PRINT  CURMAP(R0) ;Show which one it is
        RETURN
```


Real-Time Programs

```
SHODAT: .PRINT  #CNFGIS          ;Preface config value
        MOV     @#RMONST,R0      ;Pick up pointer to RMON base
        MOV     CONFIG(R0),R0    ;Get the current CONFIG value
        CALL    PRTOCT          ;Display it
        RETURN

MAPIOP: .BYTE    5,140           ;EMT arg block to map to I/O page
MAPMON: .BYTE    6,140           ;EMT arg block to map to "RMON"
AREA:   .WORD    10              ;EMT arg block
CSRSAB: .WORD    0               ;Save CSR for restoration on exit
CURMAP: .WORD    TORMON          ;Current mapping message table
        .WORD    TOIOPG
        .NLIST  BEX

TORMON: .ASCIZ   /simulated RMON./
TOIOPG: .ASCIZ   "I/O page."
MAPMSG: .ASCII   /PAR 7 is currently mapped to /<200>
CNFGIS: .ASCII   /Current value of CONFIG word in simulated RMON is /<200>
TYPE:   .ASCIZ   /Characters entered on serial line will be displayed here:/
IOPMSG: .ASCIZ   <12>"Now switching PAR 7 mapping to the I/O page."
GOBACK: .ASCIZ   <15><12>/Returning PAR 7 mapping to simulated RMON./
NOPRIV: .ASCII   /Real-time support not specified during TSGEN or /
        .ASCIZ   /user not privileged./
        .END     START
```

11.1.2 EMT to remap the program region to the simulated RMON.

The following EMT can be used to cause the virtual address mapping region of the job in the range 160000 to 177777 to be returned to normal mapping if it had previously been mapped to the I/O page. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    6,140
```

Example:

See the example program MAPIOP in the section on mapping the I/O page into the program space.

11.1.3 EMT to peek at the I/O page.

The following EMT can be used to access a word in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. (Note that the .PEEK and .POKE EMTs can also be used to access parts of the I/O page. See Chapter 14 for more information on the effects of the .PEEK and .POKE EMTs with TSX-Plus.) The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    1,140
.WORD    address
```

where "address" is the address of the word in the I/O page to be accessed. The contents of the specified word in the I/O page are returned in R0. Note that with this and other EMTs that access the I/O page, if an invalid address is specified, an error will result with the message:

?MON-F-Kernel mode trap within TSX-Plus

Example:

```
.TITLE   PEEKIO
.ENABL   LC
```

;Demonstrate TSX-Plus EMTs to peek and poke into the I/O page

```
CTRLC   = 3                ;Control-C
RMONST   = 54               ;Pointer in SYSCOM area to start of "RMON"
CONFIG   = 300              ;Fixed offset into "RMON" of config word
RCSR     = 176540           ;Serial line RCSR address
RBUF     = RCSR+2           ;Line input buffer address
```

```
.MCALL   .PRINT,.EXIT,.TTYOUT,.GVAL
.GLOBL   PRTOCT
```

```
START:   CALL    SHOMAP      ;Display current PAR 7 mapping
          CALL    SHOCON      ;Demonstrate "RMON" mapping

          MOV     #PEEKIO,R0   ;Point to EMT arg block to
          EMT     375          ;Peek into the I/O page
          BCC     1$           ;Branch if OK
          .PRINT  #NOPRIV      ;You aren't allowed to do that
          .EXIT

1$:       MOV     R0,CSRSAV     ;Save a copy of current CSR

          CLR     POKVAL        ;Want to disable interrupts
          MOV     #POKEIO,R0    ;Point to EMT arg block to
```


Real-Time Programs

```

                EMT      375                ;Poke a value into the I/O page
                ;Shouldn't be error if peek worked
2$:      .PRINT  #TYPE                ;Prompt for input from I/O page
      MOV      #PEEKIO,R0            ;Point to EMT arg block to
      EMT      375                ;Check the RCSR
      TSTB     R0                    ;Is anything available from the line
      BPL      2$                  ;No, keep checking

      ADD      #2,PEKADD            ;Point to the receiver buffer
      MOV      #PEEKIO,R0            ;Point to EMT arg block to
      EMT      375                ;Get the input character
      CMPB     R0,#CTRLC            ;Should we quit?
      BEQ      3$                  ;Quit on ^C
      .TTYOUT                                ;Display the character
      SUB      #2,PEKADD            ;Point back to the RCSR
      BR       2$                  ;And repeat

3$:      MOV     CSRSAB,POKVAL        ;Want to restore the original CSR status
      MOV      #POKEIO,R0            ;Point to EMT arg block to
      EMT      375                ;Restore the CSR

      .EXIT

SHOMAP: .PRINT  #MAPMSG                ;Current mapping preface
      .GVAL     #AREA,#-8.            ;What is our current PAR 7 mapping?
      ASL      R0                    ;Convert to word offset
      .PRINT    CURMAP(R0)            ;Show which one it is
      RETURN

SHOCON: .PRINT  #CNFGIS                ;Preface config value
      MOV      @#RMONST,R0            ;Pick up pointer to RMON base
      MOV      CONFIG(R0),R0          ;Get the current CONFIG value
      CALL     PRTOCT                ;Display it
      RETURN

PEEKIO: .BYTE   1,140                ;EMT arg block to peek into the I/O page
PEKADD: .WORD   RCSR                  ;Address to be read
POKEIO: .BYTE   2,140                ;EMT arg block to poke into the I/O page
POKADD: .WORD   RCSR                  ;Address to be modified
POKVAL: .WORD   0                    ;Word to be moved to POKADD
AREA:   .WORD   10                    ;EMT arg block
CSRSAB: .WORD   0                    ;Save CSR for restoration on exit
CURMAP: .WORD   TORMON                ;Current mapping message table
      .WORD   TOIOPG
      .NLIST   BEX

TORMON: .ASCIZ  /simulated RMON./
TOIOPG: .ASCIZ  "I/O page."
MAPMSG: .ASCII  /PAR 7 is currently mapped to /<200>
CNFGIS: .ASCII  /Current value of CONFIG word in simulated RMON is /<200>
TYPE:   .ASCIZ  /Characters entered on serial line will be displayed here:/

```



```

NOPRIV: .ASCII /Real-time support not specified during TSGEN or /
        .ASCIZ /user not privileged./
        .END   START

```

11.1.4 EMT to poke into the I/O page.

The following EMT can be used to store a value into a cell in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE    2,140
.WORD    address
.WORD    value

```

where "address" is the address of the cell in the I/O page and "value" is the value to be stored.

Example:

See the example program PEEKIO in the section on peeking into the I/O page.

11.1.5 EMT to bit-set a value into the I/O page.

The following EMT can be used to perform a bit-set (BIS) operation into a cell in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE    3,140
.WORD    address
.WORD    value

```

where "address" is the address of the cell in the I/O page and "value" is the value that will be bit-set into the cell.

Real-Time Programs

Example:

```

        .TITLE   BISIO
        .ENABL   LC

;Demonstrate TSX-Plus EMTs to bit-set and bit-clear into the I/O page

CTRLC = 3           ;Control-C
RMONST = 54          ;Pointer in SYSCOM area to start of "RMON"
INTNBL = 100         ;RCSR interrupt enable bit
CONFIG = 300         ;Fixed offset into "RMON" of config word
RCSR   = 176540      ;Serial line RCSR address
RBUF   = RCSR+2      ;Line input buffer address

        .MCALL   .PRINT,.EXIT,.TTYOUT,.GVAL

START:  CALL      SHOMAP           ;Display current PAR 7 mapping
        CALL      SHOCON          ;Demonstrate "RMON" mapping

        ;Want to clear input interrupts
        MOV       #BICIO,R0       ;Point to EMT arg block to
        EMT       375             ;Clear a bit in the I/O page
        BCC       1$              ;Branch if OK
        .PRINT    #NOPRIV         ;You aren't allowed to do that
        .EXIT

1$:     ;OK, interrupts should be disabled
        .PRINT    #TYPE           ;Prompt for input from I/O page
2$:     MOV       #PEEKIO,R0       ;Point to EMT arg block to
        EMT       375             ;Check the RCSR
        TSTB      R0              ;Is anything available from the line
        BPL       2$              ;No, keep checking

        ADD       #2,PEKADD        ;Point to the receiver buffer
        MOV       #PEEKIO,R0       ;Point to EMT arg block to
        EMT       375             ;Get the input character
        ;Value from peek is returned in R0
        CMPB      R0,#CTRLC        ;Should we quit?
        BEQ       3$              ;Quit on ^C
        .TTYOUT    ;Display the character
        SUB       #2,PEKADD        ;Point back to the RCSR
        BR        2$              ;And repeat

3$:     ;Want to set input interrupts
        MOV       #BISIO,R0        ;Point to EMT arg block to
        EMT       375             ;Set a bit in the I/O page

        .EXIT

SHOMAP: .PRINT     #MAPMSG          ;Current mapping preface
        .GVAL      #AREA,#-8.      ;What is our current PAR 7 mapping?

```


Real-Time Programs

```

ASL      R0          ;Convert to word offset
.PRINT   CURMAP(R0)  ;Show which one it is
RETURN

SHOCON: .PRINT #CNFGIS      ;Preface config value
MOV      @#RMONST,R0      ;Pick up pointer to RMON base
MOV      CONFIG(R0),R0    ;Get the current CONFIG value
CALL     PRTOCT           ;Display it
RETURN

PRTOCT: MOV      R1,-(SP)   ;Save r1-r3 on the stack
MOV      R2,-(SP)         ;R0 contains word of interest
MOV      R3,-(SP)
MOV      #EOW,R2          ;Point to end of 6 char output buffer
MOV      #6,R3            ;Set up counter for 6 chars
1$: MOV      R0,R1         ;Set up mask for low 3 bits (1s digit)
BIC      #177770,R1       ;Get low 3 bits
ADD      #'0,R1           ;Convert to ascii
MOVB     R1,-(R2)         ;Fill octal digits in from end
CLC
ROR      R0
ASH      #-2,R0           ;Shift out bits just converted
SOB      R3,1$           ;Repeat for 6 chars
.PRINT   #CHARS          ;Display result at console
MOV      (SP)+,R3         ;Restore registers r1-r3
MOV      (SP)+,R2
MOV      (SP)+,R1
RETURN

PEEKIO: .BYTE      1,140   ;EMT arg block to peek into the I/O page
PEKADD: .WORD      RCSR    ;Address to be read
BICIO:  .BYTE      4,140   ;EMT arg block to clear a bit in the I/O page
        .WORD      RCSR    ;Input status register
        .WORD      INTNBL  ;Interrupt enable mask
BISIO:  .BYTE      3,140   ;EMT arg block to set a bit in the I/O page
        .WORD      RCSR    ;Input status register
        .WORD      INTNBL  ;Interrupt enable mask
AREA:   .BLKW      10      ;EMT arg block
CSRSAB: .WORD      0       ;Save CSR for restoration on exit
CHARS:  .BLKB      6       ;6 char output buffer
EOW:    .WORD      0       ;Terminator for .PRINT
CURMAP: .WORD      TORMON   ;Current mapping message table
        .WORD      TOIOPG
        .NLIST      BEX

TORMON: .ASCIZ     /simulated RMON./
TOIOPG: .ASCIZ     "I/O page."
MAPMSG: .ASCII     /PAR 7 is currently mapped to /<200>
CNFGIS: .ASCII     /Current value of CONFIG word in simulated RMON is /<200>
TYPE:   .ASCIZ     /Characters entered on serial line will be displayed here:/
NOPRIV: .ASCII     /Real-time support not specified during TSGEN or /

```


Real-Time Programs

```
.ASCIZ  /user not privileged./  
.END    START
```

11.1.6 EMT to do a bit-clear into the I/O page.

The following EMT can be used to perform a bit-clear (BIC) operation into a cell in the I/O page without requiring the job's virtual address region to be mapped to the I/O page. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    4,140  
.WORD    address  
.WORD    value
```

where "address" is the address of the cell in the I/O page and "value" is the value to be bit-cleared into the specified cell.

Example:

See the example program BISIO in the section on doing a bit-set into the I/O page.

11.2 Mapping to a physical memory region

In certain circumstances, it is desirable to map a portion of virtual memory to a specific area in physical memory which is not in the I/O page. Possible examples would be ROM memory or an array processor. This mapping is done by altering one or more of the Page Address Registers (PARs) for the job. Each PAR maps 8192 bytes of memory from the virtual job space into physical memory. There are 8 PAR's ($8 \times 8192 = 64\text{Kb}$). The region of memory mapped through each PAR is shown by the table below:

PAR	Virtual Region
---	-----
0	000000 - 017777
1	020000 - 037777
2	040000 - 057777
3	060000 - 077777
4	100000 - 117777
5	120000 - 137777
6	140000 - 157777
7	160000 - 177777

The form of this EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE 17,140
.WORD par-number
.WORD phys-address
.WORD size
.WORD access
```

"par-number" is the number of the Page Address Register (PAR) that corresponds to the beginning of the program virtual address region that is to be mapped.

"phys-address" is the physical address to which the virtual address is to be mapped. The physical address is specified as an address divided by 64 (decimal). That is, the physical address represents the 64-byte block number of the start of the physical region. Note that the physical address of any 64-byte block within a 22-bit physical address space can be represented in 16 bits.

"size" is the number of 64-byte blocks of memory to be mapped through the virtual region. Each PAR can map up to 128 64-byte blocks of memory. If more than 128 blocks are mapped, successively higher PARs are set up to map the remainder of the region.

"access" indicates if the mapped region is to be allowed read-only access or both read and write access: 0 = read-only; 1 = read/write.

This EMT may be used to map any of the PARs to any physical address regions desired (even to map PAR 7 to the I/O page). The use of this EMT does not affect mapping set up previously for other PARs. If the "size" parameter is 0 (zero), all PAR mapping is reset and the normal virtual address mapping for the job is restored.

Note that this EMT is not equivalent to the extended memory EMTs used with PLAS (Program's Logical Address Space) requests and virtual overlays and virtual arrays. See Chapter 8 for a discussion of job environment and the appropriate RT-11 manuals for a more complete discussion of PLAS features.

Real-Time Programs

Example:

```
.TITLE  MAPPHY
.ENABL  LC

; Demonstrate TSX-Plus EMT to map to any physical address

CTRLC  = 3                ;Control-C
INTNBL  = 100             ;Interrupt enable bit
RCSR    = 176540          ;Serial line RCSR address
RBUF    = RCSR+2          ;Line input buffer address

.MCALL  .PRINT,.EXIT,.TTYOUT,.GVAL

START:  .PRINT  #MAPMSG    ;Say we are switching to physical mapping
        MOV     #MAPPHY,RO ;Point to EMT arg block to
        EMT     375        ;Map to physical memory
        BCC     1$         ;Branch if OK to map
        .PRINT  #NOPRIV    ;You aren't allowed to do that
        .EXIT

1$:     .PRINT  #TYPE       ;Prompt for input from I/O page
        BIC     #INTNBL,@#RCSR ;Disable interrupts on serial line
2$:     TSTB    @#RCSR      ;Is anything available from the line
        BPL     2$         ;No, keep checking
        MOVB    @#RBUF,RO   ;Get the new character
        CMPB    RO,#CTRLC   ;Should we quit?
        BEQ     3$         ;Quit on ^C
        .TTYOUT ;Display the character
        BR      2$         ;And repeat

3$:     CLR     SIZE        ;If .size is 0, original mapping is restored
        MOV     #MAPPHY,RO ;Point to EMT arg block to
        EMT     375        ;Revoke mapping to physical address

        .EXIT

MAPPHY: .BYTE    17,140     ;EMT arg block to map to physical memory
        .WORD    7         ;remap PAR 7
        .WORD    177600    ;17760000/64. address to map into PAR 7
SIZE:   .WORD    200       ;size of region to be mapped - full 8kb
        .WORD    1         ;access 0=read-only, 1=read/write
        .NLIST  BEX

MAPMSG: .ASCIZ   /PAR 7 mapping is now directed to top of physical memory./
TYPE:   .ASCIZ   /Characters entered on serial line will be displayed here:/
NOPRIV: .ASCII   /Real-time support not specified during TSGEN or /
        .ASCIZ   /user not privileged./
        .END     START
```


11.3 Requesting exclusive system control

The following EMT allows real-time jobs to gain exclusive access to the system while they perform time-critical tasks. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 14,140

The effect of this EMT is to cause the TSX-Plus job scheduler to ignore all other jobs, even higher priority jobs - including fixed-high-priority jobs, until the real-time job relinquishes exclusive access control. Note that this is different (and more powerful) than giving the real-time job a higher execution priority because all other jobs are completely prevented from executing even if the real-time job goes into a wait state causing the CPU to become idle.

The form of the EMT used to relinquish exclusive system access is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 15,140

The following restrictions apply to a job that has issued an exclusive access EMT:

1. The job is automatically locked in memory during the time it has exclusive access to the system. If you wish to use the TSX-Plus EMT that locks a job in the lowest portion of memory, that EMT must be executed before calling this EMT to gain exclusive access to the system.
2. The size of the job may not be changed while it has exclusive access to the system.

Exclusive access is automatically relinquished if the job exits or traps but is retained if the job chains to another job.

Real-Time Programs

Example:

```
.TITLE  STEALS
.ENABL  LC

; Demonstrate TSX-Plus EMT to obtain exclusive system control

LEADIN  =      35          ;TSX-Plus program controlled terminal
                                ;option lead-in character.
JSW      =      44          ;Job status word address
TTSPC    =     10000        ;TT special mode bit in JSW
NOWAIT   =     100          ;TT nowait bit in JSW

.MCALL  .PRINT,.TTYOUT,.TTYIN,.EXIT

START:  MOV      #TTSPC!NOWAIT,@#JSW      ;Set TT special mode and nowait
                                              ;bits in the JSW
        .TTYOUT  #LEADIN                  ;And make TSX-Plus understand both
        .TTYOUT  #`S
        .TTYOUT  #LEADIN
        .TTYOUT  #`U

        .PRINT   #MSG                      ;Prompt for input
        MOV      #STEALS,R0                ;Point to EMT arg block to
        EMT      375                      ;Steal exclusive system control
; . . .                                     ;Time critical processing goes here
        .TTYIN   ;Simulated here with terminal input
        MOV      #LETGO,R0                ;Point to EMT arg block to
        EMT      375                      ;Relinquish exclusive system control
        .EXIT

        .NLIST   BEX

STEALS:  .BYTE    14,140                  ;EMT arg block for exclusive system control
LETGO:   .BYTE    15,140                  ;EMT arg block to relinquish exclusive control
MSG:     .ASCII   /Other users are locked out until you press a key:/<200>

.END     START
```

11.4 Locking a job in memory

In time-critical real-time applications where a program must respond to an interrupt with minimum delay, it may be necessary for the job to lock itself in memory to avoid program swapping. This facility should be used with caution since if a number of large programs are locked in memory there may not be enough space left to run other programs.

TSX-Plus provides two program locking facilities. The first moves the program to the low end of memory before locking it; this is done to avoid fragmenting

available free memory. This type of lock should be done if the program is going to remain locked in memory for a long period of time. However, this form of locking is relatively slow since it may involve program swapping. The second locking facility simply locks the program into the memory space it is occupying when the EMT is executed without doing any repositioning. This EMT has the advantage that it is extremely fast but free memory space may be non-contiguous.

The form of the EMT used to lock a program in low memory (re-positioning if necessary) is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    7,140
```

The form of the EMT used to lock a job in memory without repositioning it is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    13,140
```

Example:

See the example program ATTVEC in the section on connecting a real-time interrupt to a completion routine.

11.5 Unlocking a job from memory

When a job locks itself in memory, it remains locked until the job exits or the following EMT is executed. The form of the EMT used to unlock a job from memory is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    10,140
```

Example:

See the example program ATTVEC in the section on connecting a real-time interrupt to a completion routine.

Real-Time Programs

11.6 Suspending/Resuming program execution

The RT-11 standard .SPND and .RSUM EMTs are used by TSX-Plus real-time jobs to suspend and resume their execution. Frequently, a real-time job will begin its execution by connecting interrupts to completion routines and doing other initialization and then will suspend its execution while waiting for a device interrupt to occur.

The .SPND EMT causes the main-line code in the job to be suspended. Completion routines are not affected and execute when interrupts occur. If a completion routine executes a .RSUM EMT, the main-line code will continue execution at the point following the .SPND when the completion routine exits. Refer to the RT-11 Programmer's Reference Manual for further information about the use of .SPND and .RSUM.

Example:

See the example program ATTVEC in the section on connecting a real-time interrupt to a completion routine.

11.7 Converting a virtual address to a physical address

When controlling devices that do direct memory access (DMA), it is necessary to be able to obtain the physical memory address (22-bit) that corresponds to a virtual address in the job. Note that a job should lock itself in memory before performing this EMT. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

```
.BYTE    0,140
.WORD    virtual-address
.WORD    result-buffer
```

where "virtual-address" is the virtual address that is to be converted to a physical address and "result-buffer" is the address of a two word area that is to receive the physical address. The low-order 16 bits of the physical address are stored in the first word of the result buffer and the high-order 6 bits of the physical address are stored in bit positions 4-9 of the second word of the result buffer.

Example:

```
.TITLE PHYADD
.ENABL LC
```

```
; Demonstrate TSX-Plus EMT to convert a virtual to a physical address
```

```
.MCALL .PRINT,.EXIT
```

```
START: MOV    #LOKJOB,R0    ;Point to EMT arg block to
      EMT      375          ;Lock job in memory
      BCC      1$          ;Continue if OK
      .PRINT   #NOPRIV      ;Explain the problem
      BR       2$          ;and quit
1$:    .PRINT   #STRADD      ;Say where this program was loaded
      MOV      #START,R0    ;Get the virtual address
      CALL     PRTADD        ;and display it to the terminal
2$:    .EXIT

PRTADD: MOV     R0,VIRADD    ;Put the virtual address into the arg block
      MOV      #PHYADD,R0   ;Point to EMT arg block to
      EMT      375          ;Convert virtual to physical address
      MOV      BUFFER,R1    ;Retrieve the low address
      MOV      <BUFFER+2>,R2 ;And the high address
      BIC      #^C1760,R2   ;Use only bits 4-9
      MOV      #4,R3        ;Throw away bits 0-3
1$:    ASR      R2
      SOB      R3,1$
      MOV      #CHREND,R4   ;Point to end of character output buffer
      MOV      #8.,R3       ;Need to convert 8 digits
2$:    MOV      R1,R0        ;Get a copy of low bits into R0
      BIC      #^C7,R0      ;Mask out all but low digit
      ADD      #^0,R0       ;Convert to ASCII
      MOVB     R0,-(R4)      ;Put low digit in output buffer
      MOV      #3,R5        ;Shift over 3 bits to next digit
3$:    ASR      R2           ;Low bit of high address
      ROR      R1           ;into high bit of low address
      SOB      R5,3$
      SOB      R3,2$        ;Do all 8 digits
      .PRINT   R4           ;Print out the result
      RETURN

LOKJOB: .BYTE   13,140      ;EMT arg block to lock job in memory
PHYADD: .BYTE   0,140       ;EMT arg block to determine physical address
VIRADD: .WORD   0           ;Virtual address to be located
      .WORD    BUFFER       ;Location of 2 word result buffer
BUFFER: .WORD   0,0         ;Will hold low and high physical address
      .BLKB    10.          ;Buffer to hold ASCII value of address
CHREND: .WORD   0           ;End of ASCII buffer
```


Real-Time Programs

```
.NLIST BEX
NOPRIV: .ASCII /Real-time not included in TSGEN or user /
        .ASCIZ /not privileged./
STRADD: .ASCII /This program is loaded at START = /<200>
        .END START
```

11.8 Specifying a program-abort device reset list

The standard RT-11 .DEVICE EMT is used by TSX-Plus real-time jobs to specify a list of device control registers to be loaded with specified values when the job terminates. This feature is useful in allowing real-time devices to be turned-off if the real-time control program aborts. The TSX-Plus .DEVICE EMT has the same form and options as the standard RT-11 .DEVICE EMT. See the RT-11 Programmer's Reference Manual for further information.

The connection between interrupt vectors and interrupt service routines or interrupt completion routines is automatically dropped when a job terminates.

11.9 Setting processor priority level

The following EMT allows a program to set the processor priority level. This can be useful in a situation where it is necessary for a real-time job to block interrupts for a short period of time while it is performing some critical operation. On return from this EMT, the job is executing in user mode with the specified processor priority level. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following EMT argument block:

```
.BYTE    16,140
.WORD    prio-level
```

where "prio-level" should be 7 to cause interrupts to be disabled or 0 to reenable interrupts. Interrupts should not be disabled for a long period of time or clock interrupts and terminal I/O interrupts will be lost.

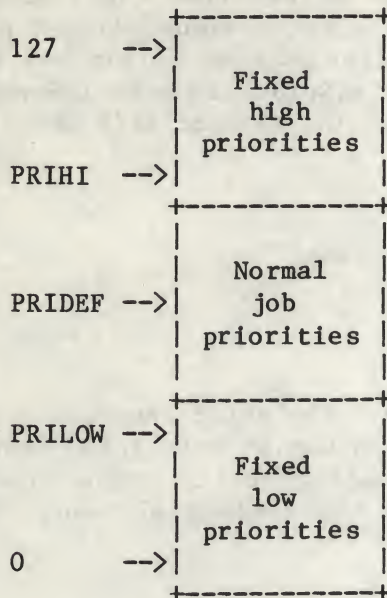
Note that it is not possible to raise the processor priority level by storing directly into the processor status word from a real-time program because the PDP-11 hardware disallows modification of the PSW by a user mode program even if the program has access to the I/O page.

Example:

See the example program ATTVEC in the section on connecting a real-time interrupt to a completion routine.

11.10 Setting Job Execution Priority

Jobs may be assigned priority values in the range 0 to 127 to control their execution scheduling relative to other jobs. The priority values are arranged in three groups: the fixed-low-priority group consists of priority values from 0 up to the value specified by the PRILOW sysgen parameter; the fixed-high-priority group ranges from the value specified for the PRIHI sysgen parameter up to 127; the middle priority group ranges from (PRILOW+1) to (PRIHI-1). The current values for PRIHI and PRILOW may be determined with the SHOW PRIORITY command, or from within a program with a .GVAL request. The following diagram illustrates the priority groups:



Job scheduling is performed differently for jobs in the fixed-high-priority and fixed-low-priority groups than for jobs with normal interactive priorities. Jobs with priorities in the fixed-low-priority group (0 to PRILOW) and the fixed-high-priority group (PRIHI to 127) execute at fixed priority values. That is, the priority absolutely controls the scheduling of the job for execution relative to other jobs. A job with a fixed priority is allowed to execute as long as it wishes until a higher priority job becomes active.

The fixed-high-priority group is intended for use by real-time programs. The fixed-low-priority group is intended for use by very low priority background tasks. Normal time-sharing jobs should not be assigned priorities in either of the fixed priority groups.

The middle group of priorities from (PRILOW+1) to (PRIHI-1) are intended to be used by normal, interactive, time-sharing jobs. Jobs with these assigned priorities are scheduled in a more sophisticated manner than the fixed-priority jobs. In addition to the assigned priority, external events such as terminal input completion, I/O completion, and timer quantum expiration play a role in determining the effective scheduling priority.

Real-Time Programs

When a job with a normal priority switches to a virtual line, the priority of the disconnected job is reduced by the amount specified by the PRIVIR sysgen parameter. This causes jobs that are not connected to terminals to execute at a lower priority than jobs that are. This priority reduction does not apply to jobs with priorities in the fixed-high-priority group or the fixed-low-priority group. The priority reduction is also constrained so that the priority of jobs in the normal job priority range will never be reduced below the value of (PRILOW+1).

The following EMT can be used to set the job priority from within a program. Unlike the other real-time EMT's in this chapter, this EMT does not require operator privilege. The maximum priority which may be used by a job is set by the system manager. The job priority can also be set from the keyboard with the SET PRIORITY command. The current job priority, maximum allowed priority, and fixed-high- and fixed-low-priority boundaries may be determined with the .GVAL request. See the TSX-Plus System Manager's Guide for more information on the significance of priority in job scheduling. The form of this EMT is:

EMT 375

with R0 pointing to the following EMT argument block:

```
.BYTE    0,150
.WORD    value
```

where "value" is the priority value for the job. The valid range of priorities is 0 to 127 (decimal). The maximum job priority may be restricted through the logon mechanism. If a job attempts to set its priority above its maximum allowed priority, its priority will be set to the maximum allowed. This EMT does not return any errors.

Example:

See Chapter 7 for an example of the use of this EMT.

11.11 Connecting interrupts to real-time jobs

One of the most important uses for real-time jobs is to service interrupts for non-standard devices. TSX-Plus provides three mechanisms for connecting user-written real-time programs to interrupts: device handlers, interrupt service routines, and interrupt completion routines. Interrupt service routines and interrupt completion routines permit jobs to process interrupts without the necessity of writing a special device handler. However, there are restrictions on the use of these two methods which must be considered when deciding the appropriate method for handling special device interrupts.

The fastest method of handling interrupts is to write a device handler (driver). Device handlers execute in kernel mode and provide the fastest possible response to interrupts. A device handler is the best choice when interrupts occur at a rate which significantly loads the system. Device

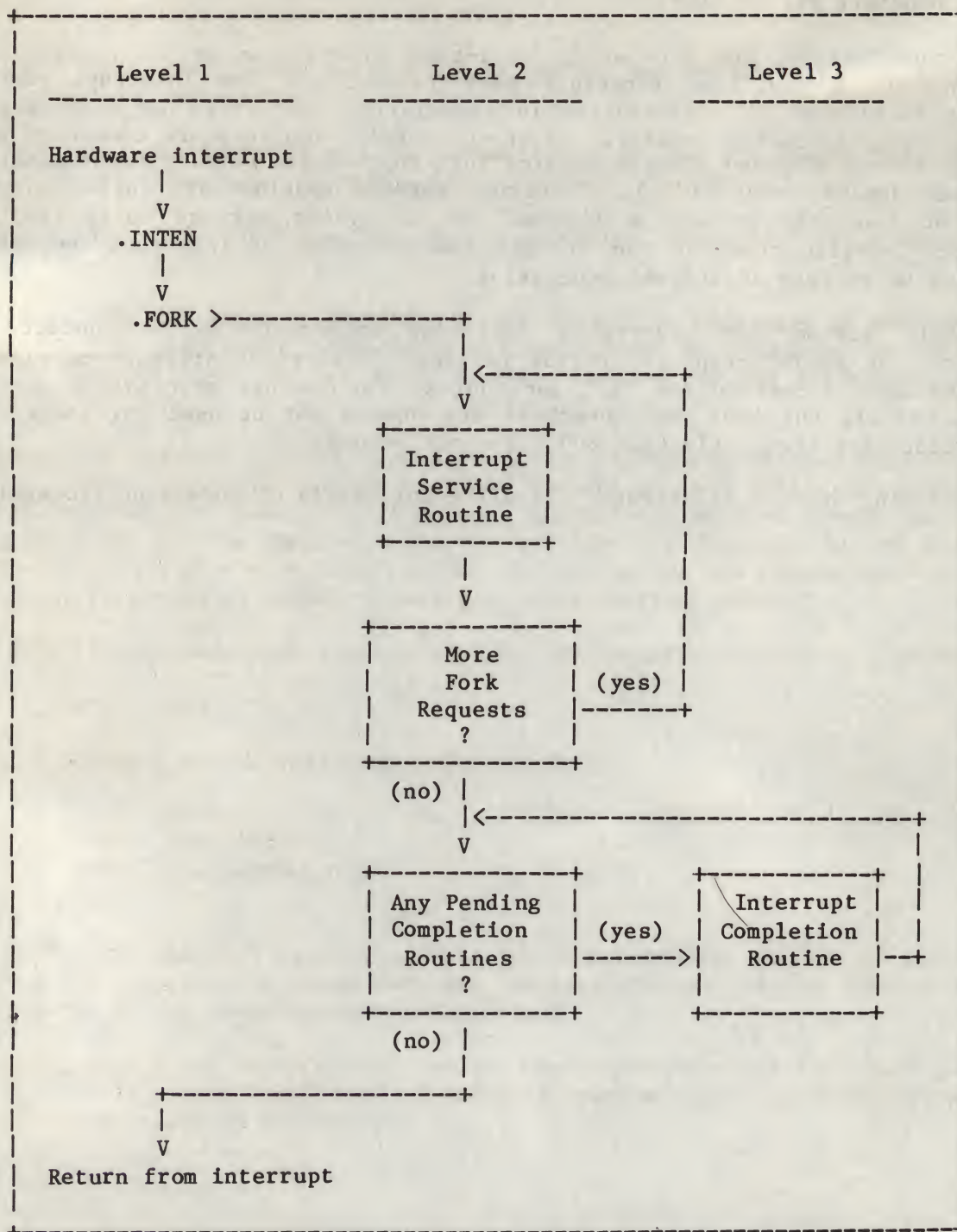
handlers written for use with TSX-Plus should conform to the rules for writing device handlers for RT-11XM.

The second method for processing real-time interrupts is to connect the interrupt to a real-time interrupt service routine. The interrupt service routine is written as a subroutine in a real-time job; it is not necessary to write a separate device handler. Interrupt service routines are connected with minimal system overhead and can service interrupts quite rapidly (approximately 2000 per second on an 11/44). Interrupt service routines are called in user mode but can only execute a limited set of system service calls (EMT's). Interrupt service routines can trigger the execution of interrupt completion routines to perform additional processing.

The third method for processing real-time interrupts is to connect the interrupt to an interrupt completion routine. Real-time interrupt completion routines have access to the full job context and can use most system service calls (EMT's), but have more overhead and should not be used for interrupts that occur more frequently than 200 times per second.

The following diagram illustrates the different levels of interrupt processing.

Interrupt Processing



This diagram shows that there are three "levels" of interrupt processing. Level 1 is entered when a hardware interrupt occurs. In this level the processor (hardware) priority is set to 7 which causes other interrupt requests to be temporarily blocked. After some brief interrupt entry processing, the system performs a .FORK operation which queues a request for processing at fork level and then drops the processor priority to 0. At this time another hardware interrupt can occur, in which case the cycle will be repeated and another request for fork level processing will be placed on the queue. Note that if .FORK requests are issued at a sustained high rate, such that numerous prior requests cannot be serviced, a system error may eventually occur.

Level 2 processing is also known as "fork level" processing. This level of interrupt processing services requests that were placed on a queue by the .FORK operation. Hardware interrupts are enabled during this processing and if any other interrupts occur their .FORK requests are placed at the end of the queue. Interrupt service requests are processed serially in the order that the interrupts occurred. Only two system service calls may be used from service routines running at fork level: a request to queue a user completion routine for subsequent processing; and the .RSUM EMT. Fork level processing also has associated priority levels. Real-time processing has the highest priority. For more information on prioritized .FORK requests, see the TSX-Plus System Manager's Guide.

Level 3 processing occurs in "job state". That is, the TSX-Plus job execution scheduler selects the highest priority job or completion routine and passes execution to it. Completion routines run with full job context and may issue system service calls (except USR calls) as needed. Completion routines are serialized for each job. That is, all other completion routines (including higher priority interrupt completion routines) which are scheduled for the same job are queued for execution and will not be entered until the current completion routine exits. During level 3 processing, interrupts are enabled and job execution may be interrupted to process fork level interrupt service routines or by higher priority completion routines for other jobs.

11.11.1 Interrupt service routines.

Interrupt service routines execute in user mode but require a minimal amount of system overhead. When an interrupt is received through a vector which has been connected to an interrupt service routine, TSX-Plus executes a .INTEN and a .FORK, sets up memory management for the appropriate job, saves the status of the floating point unit (FPU) if it is in use, and passes control to the interrupt service routine. Using this approach, interrupts may be serviced at about 2000 per second on a PDP-11/44. Lower rates should be expected on slower processors.

Several restrictions apply to this method of interrupt processing:

- a) The job must be locked in memory before connecting to the interrupt vector and must remain locked in memory as long as the interrupt connection is in effect.

Real-Time Programs

- b) The processing done by the interrupt service routine should be brief since other interrupts that do .FORKS will be queued until the interrupt service routine exits.
- c) Only two system service calls (EMTs) are valid within a interrupt service routine:
 - 1) A .RSUM EMT may be issued to cause the job's main-line code to continue processing if it has done a .SPND.
 - 2) The TSX-Plus EMT which schedules execution of a completion routine.

Access to the I/O page is possible if the main-line code sets up such mapping prior to the interrupt. Registers are undefined on entry to an interrupt service routine, and do not have to be preserved by the interrupt service routine.

An interrupt service routine must exit with a RETURN instruction (RTS PC), not an RTI. Since interrupt service routines execute at fork level, job scheduling is not relevant for them. All fork level processing, whether queued for system processing or for an interrupt service routine, is executed in the order in which the interrupts were received and execute before any completion routines, fixed-priority jobs or normal interactive time-sharing jobs.

The form of the request to connect an interrupt service routine to a vector is:

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 20,140
.WORD vector-address
.WORD service-routine
.WORD 0
```

where "vector-address" is the address of the interrupt vector to which the service routine is to be connected, and "service-routine" is the address of the entry point of the interrupt service routine.

The total number of vectors that can be connected either to interrupt service routines or interrupt completion routines is determined by the RTVECT parameter during TSX-Plus system generation.

Error:

Code	Meaning
-----	-----
0	Real time not included or job not privileged
1	Maximum number of vectors already in use
2	Some other job is already connected to that vector
3	Job is not locked in memory

The association between an interrupt service routine and an interrupt vector may be released by the same EMT as used to disconnect an interrupt completion routine. See the section on releasing an interrupt connection later in this chapter.

Example:

```
.TITLE INTSVC
.ENABL LC
```

```
; Demonstrate EMT to implement an interrupt service routine
; Simple file capture program. Steal a time-sharing line
; and put everything that comes in on it into a file.
```

```
.MCALL .TTYIN,.TTYOUT,.EXIT,.PEEK,.POKE,.GVAL,.SCCA
.MCALL .ENTER,.LOOKUP,.CSISPC,.WRITE,.PURGE,.CLOSE,.DEVICE
```

```
VECTOR = 330           ;Serial line vector
RCSR    = 176530        ;Serial line RCSR address
RBUF    = RCSR+2        ;Serial input buffer address
BUFSIZ  = 256.          ;Size of buffer in words
INTNBL  = 100           ;Interrupt enable bit in RCSR
CTRLC   = 3             ;ASCII control-C
CTRLD   = 4             ;ASCII control-D
```

```
START:  .GVAL  #AREA,#-6. ;See if job is privileged
        TST    R0         ;1 if priv, 0 if not
        BEQ    QUIT       ;Immediate exit if not priv
        MOV    #LOKJOB,R0 ;Point to EMT arg block to
        EMT    375        ;Lock job in memory
        MOV    #MAPIO,R0  ;Point to EMT arg block to
        EMT    375        ;Map PAR7 into the I/O page
```

```
GETFIL: MOV    SP,SPSAVE  ;Save SP prior to CSI call
1$:      .CSISPC #OUTSPC,#DEFEXT,#0 ;Get file specification
        BCS    1$         ;Repeat until valid command line
        MOV    SPSAVE,SP  ;Restore SP, no valid switches
        .ENTER #AREA,#0,#INSPC,#-1 ;Open largest possible output file
        BCS    1$         ;On error, ask for new file
        MOV    #BUFF1,BUFPTR ;Initialize buffer pointer
        CLR    BLOCK      ;Initialize output file block count
```


Real-Time Programs

```

; Set up to accept terminal commands
;  ctrl-c, ctrl-c to abort; ctrl-d to end transmission
        .SCCA    #AREA,#TTSTAT    ;Disable control-c abort
        MOV      #ACTCTD,R0       ;Point to EMT arg block to
        EMT      375              ;Activate input on ^D

; Since we are going to borrow a current time-sharing line, we need
; to save its vector pointers for later restoration
        MOV      @#RCSR,CSRSAV    ;Save old status bits, esp. int. enable
        .DEVICE  #AREA,#DEVLST   ;Force restoration of RCSR on exit
        BIC      #INTNBL,@#RCSR   ;Disable interrupts until ready
        .PEEK    #AREA,#VECTOR    ;Get normal vector pointer
        MOV      R0,VECSAV        ;And save it for later restoration
        .PEEK    #AREA,#VECTOR+2  ;Get normal priority
        MOV      R0,VECSV2        ;And save it for later restoration

; Now attach interrupt service routine to input vector
        MOV      #INTSVC,R0       ;Point to EMT arg block to
        EMT      375              ;Schedule interrupt service routine
GO:      BIS      #INTNBL,@#RCSR   ;Enable interrupts and wait for input
        .TTYIN   INBUF            ;Wait for terminal command during transfer

; Resume here when transfer is complete, or want to abort
WHOA:    BIC      #INTNBL,@#RCSR   ;Disable interrupts (data lost if mistake)
        CMPB     INBUF,#CTRLC     ;Was it a control-c?
        BNE      5$              ;Branch if not
        .PURGE   #0               ;If CTRL-C, throw away input
        TST      TTSTAT           ;Did we get double control-c's?
        BEQ      GETFIL          ;If not, get another file name
        BR       QUIT            ;If double control-c, then abort
5$:      CMPB     INBUF,#CTRLD     ;Is the transmission done?
        BNE      GO              ;Branch if not
        CALL     FINISH           ;Write out remainder of current buffer
        .CLOSE   #0              ;If done, close out file

; Done or abort, restore time-sharing line conditions
QUIT:    MOV      #RELVEC,R0       ;Point to EMT arg block to
        EMT      375              ;Release interrupt connection
        .POKE    #AREA,#VECTOR+2,VECSV2 ;Restore old priority
        .POKE    #AREA,#VECTOR,VECSAV  ;And old vector pointer
        .EXIT                                ;And done!

; Interrupt service routine, executes at .FORK level
ISR:     MOVB     @#RBUF,@BUFPTR   ;Put char in buffer
        INC      BUFPTR           ;And point to next location
        CMP      BUFPTR,#BUFF2    ;Which buffer in use?
        BHI      1$              ;Branch if already in second
        BLO      9$              ;Return if first not full
        MOV      #BUFF1,WRTPTR    ;Exact end of buffer 1, point to it

```


Real-Time Programs

```

1$: BR      2$          ;Schedule write
    CMP     BUFPTR,#BUFEND ;Second buffer full yet?
    BLO     9$          ;Not yet, return
    MOV     #BUFF1,BUFPTR ;Second buffer full, reset pointer
    MOV     #BUFF2,WRTPTR ;Point to second buffer for write

2$: MOV     #SCHWRT,R0    ;Point to EMT arg block to
    EMT     375          ;Schedule completion routine
                        ;to write buffer to file

9$: TST     TTSTAT       ;Does mainline want to quit? (^C^C)
    BEQ     10$          ;Branch if not
    BIC     #INTNBL,@#RCSR ;If so, disable interrupts
10$: RETURN            ;Wait for another char
                        ;Note RTS PC, not RTI !

```

; Completion routine to save buffer to file

```

CMPRTN: .WRITE #AREA,#0,R1,#256.,BLOCK ;Write buffer to file
        BCC     1$          ;Br if no error
        MOVB    CTRLC,INBUF ;On error, set abort flags
        MOV     #-1,TTSTAT
        JMP     WHOA        ;Skip .TTYIN, and abort
1$: INC     BLOCK          ;Point to next output block
    RETURN

```

; Routine to complete write of last block

```

FINISH: MOV     BUFPTR,R0    ;Get current buffer pointer
        MOV     #BUFF2,R1    ;Point to end of first buffer
        CMP     R0,R1        ;See which buffer in use
        BLO     1$          ;Skip if in first
        ADD     #2*BUFSIZ,R1 ;If in second, point to end
1$: CLRB      (R0)+          ;Zero out buffer
        CMP     R0,R1        ;All the way to the end
        BLO     1$          ;Now point back to buffer beginning
        SUB     #2*BUFSIZ,R1 ;Call write routine (NOT AS COMPLETION)
        CALL    CMPRTN
        RETURN

```

; EMT arg block areas

```

AREA:   .BLKW    10          ;General EMT arg block

LOKJOB: .BYTE    13,140      ;EMT arg block to lock job in mem

MAPIO:  .BYTE    5,140       ;EMT arg block to map I/O page to PAR 7

ACTCTD: .BYTE    0,152       ;EMT arg block to
        .WORD    ^D          ;Set activation character

```


Real-Time Programs

	.WORD	CTRLD	;Control-D
INTSVC:	.BYTE	20,140	;EMT arg block to set up interrupt svc routine
	.WORD	VECTOR	;Vector to attach
	.WORD	ISR	;Address of Interrupt Service Routine
	.WORD	0	;Required
SCHWRT:	.BYTE	21,140	;EMT arg block to sched compl routine
	.WORD	CMPRTN	;Write buffer contents to file
	.WORD	7	;Real-time priority (adds to PRIHI)
WRTPTR:	.WORD	0	;Passed in R1 to compl routine (buffer addr)
	.WORD	0	;Required
RELVEC:	.BYTE	12,140	;EMT arg block to release interrupt connection
	.WORD	VECTOR	;Vector to be released
; General storage			
OUTSPC:	.BLKW	15.	;CSI output file specs
INSPC:	.BLKW	24.	;CSI input file specs
DEFEXT:	.WORD	0,0,0,0	;No default file specs
SPSAVE:	.WORD	0	;Save stack pointer during CSI call
BLOCK:	.WORD	0	;Output file block counter
TTSTAT:	.WORD	0	;SCCA terminal status word
VECSAV:	.WORD	0	;Save original vector contents
VECSV2:	.WORD	0	; and priority
DEVLST:	.WORD	RCSR	;.DEVICE restoration list
CSRSVAV:	.WORD	0	;To hold original value of RCSR
	.WORD	0	;End of list
BUFPTR:	.WORD	0	;Current input char pointer
INBUF:	.BYTE	0,0,0,0	;Terminal command buffer
BUFF1:	.BLKW	BUFSIZ	;1 block input buffer
BUFF2:	.BLKW	BUFSIZ	;Second buffer
BUFEND:			
	.END	START	

11.11.2 Interrupt completion routines.

The TSX-Plus real-time support facility allows a program to connect a real-time interrupt to a completion routine. If this is done, TSX-Plus schedules the completion routine to be executed each time the specified interrupt occurs.

Interrupt completion routines have much greater flexibility than interrupt service routines, but require more overhead and are capable of servicing interrupts only at a lower rate. Interrupt completion routines run with full job context. This allows them to use all system service calls (except to the USR). Registers will be preserved between calls to the completion routine.

Real-time completion routines can service interrupts at rates up to about 200 interrupts per second. Devices which interrupt at a faster rate should be connected through an interrupt service routine or through a special device handler.

The total number of interrupt vectors that can be connected either to interrupt completion routines or interrupt service routines is determined by the RTVECT parameter during TSX-Plus system generation. It is possible for several interrupts to be connected to the same completion routine in a job but it is illegal for more than one job to try to connect to the same interrupt. When an interrupt completion routine is entered, R0 contains the address of the interrupt vector that caused the completion routine to be executed.

Real-time completion routines, whether directly connected to an interrupt or scheduled with the EMT for that purpose, have an associated job priority. These are software priorities, not hardware priorities; all completion routines are synchronized with the job and execute at hardware priority level 0. Completion routine priorities 1 and larger are classified as "real-time" priorities and are added to the system parameter PRIHI to yield the job execution priority, unless the resultant priority would exceed 127 in which case the priority will be 127. These completion routines will then be scheduled for execution whenever there are no executable jobs with a higher priority.

A real-time completion routine for one job will be suspended if an interrupt occurs which causes a higher priority completion routine to be queued for another job. However, a real-time completion routine for one job will never be interrupted by a completion routine for that same job regardless of the subsequent completion routine's priority. If additional requests are made to trigger the same or different completion routines while a completion routine is executing, the requests are queued for the job and are serviced in order based on their priority and the order in which they were queued.

A real-time completion routine is allowed to run continuously until one of the following events occurs: 1) the completion routine finishes execution and returns; 2) a higher-priority completion routine from another job interrupts its execution -- it is re-entered when the higher-priority routine exits; or 3) the completion routine enters a system wait state such as waiting for an I/O operation to complete or waiting for a timed interval. If a real-time completion routine enters a wait state, it returns to the same real-time priority when the wait condition completes.

Real-time completion routines with a priority of 0 are treated slightly differently than those with priorities greater than zero. The action taken depends on the priority of the main-line job when the completion routine is scheduled. If the base priority is equal to or greater than PRIHI, then the completion routine is treated just as those with real-time priorities greater than zero. That is, the completion routine is assigned a priority of PRIHI (PRIHI + the real-time priority of 0). On the other hand, if the base job priority is less than PRIHI, then the job is scheduled as a very high priority normal (non-interactive) job. The effect of this is that completion routines with a real-time priority of 0 and a base job priority less than PRIHI will interrupt normal time-sharing jobs, but are time-sliced in the normal fashion and lose their high priority if they execute longer than the system parameter QUAN1A.

Real-Time Programs

Jobs that have real-time, interrupt completion routines need not necessarily be locked in memory. If an interrupt occurs while the job is swapped out of memory, it is scheduled for execution like any other job and swapped in before the completion routine is executed. Note, however, that this condition is not optimal for timely servicing of interrupts.

When a real-time interrupt occurs, a request is placed in a queue to execute the appropriate completion routine. If the interrupt occurs again before the completion routine is entered, another request is placed in the queue so the completion routine will be invoked twice. A TSX-Plus fatal system error occurs if an interrupt occurs and there are no free completion routine request queue entries.

When a real-time completion routine completes its execution, it must exit by use of a RETURN instruction (RTS PC), not an RTI.

The form of the EMT used to connect an interrupt to a real-time completion routine is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    11,140
.WORD    interrupt-vector
.WORD    completion-routine
.WORD    priority
```

where "interrupt-vector" is the address of the interrupt vector, "completion-routine" is the address of the completion routine, and "priority" is the execution priority for the completion routine.

Error:

Code	Meaning
----	-----
0	Real-time support not included or job not privileged
1	Maximum number of vectors already in use
2	Some other job is already connected to that vector
3	Job is not locked in memory

Example:

```
.TITLE  ATTVEC
.ENABL  LC
```

; Demonstrate TSX-Plus EMTs to attach a completion routine to an interrupt

```
CTRLC = 3           ;Control-C
ERRBYT = 52
RCSR = 176540       ;Serial line RCSR address
```



```

RBUF      = RCSR+2                ;Line input buffer address

        .MCALL .PRINT,.EXIT,.TTYOUT,.SPND,.RSUM

START:  MOV      #MAPIOP,R0        ;Point to EMT arg block to
      EMT      375                ;Map to I/O page
      BCC      1$                 ;Branch if OK to map
      .PRINT   #NOPRIV            ;You aren't allowed to do that
      .EXIT

;+
;If this job were to be resident for a long time, it would be better
;to lock into low memory to avoid memory fragmentation by job swapping.
;However, since this job may have to be swapped now to get into low
;memory, locking into low memory takes longer to execute.
;
1$:      MOV      #LOKLLOW,R0      ;Point to EMT arg block to lock into low mem.
;-
1$:      MOV      #LOKJOB,R0      ;Point to EMT arg block to
      EMT      375                ;Lock the job in memory

      MOV      #ATTVEC,R0        ;Point to EMT arg block to
      EMT      375                ;Attach to an interrupt vector
      BCC      2$                 ;Continue if OK
      .PRINT   #BADATT           ;Notify can't attach to interrupt
      MOVB     @#ERRBYT,R0       ;Find out which error
      ASL      R0                 ;Convert to word offset
      .PRINT   ATTERR(R0)        ;And explain reason for error
      BR       3$                 ;Go on to unlock and exit

2$:      .PRINT   #TYPE           ;Prompt for input from interrupting device
      .SPND                      ;Now wait for an interrupt

; . . .

      MOV      #RELVEC,R0        ;Resume here on exit from completion code
      EMT      375                ;Point to EMT arg block to
                                   ;Release an interrupt vector

;+
; NOTE: Any interrupts through this vector after it has been released
;       will cause a TSX-Plus fatal system error - Unexpected Interrupt.
;-
3$:      MOV      #UNLOKJ,R0      ;Point to EMT arg block to
      EMT      375                ;Unlock this job from memory
      .EXIT

;+
; The following code will be executed as a completion routine
; when the device attached to the vector interrupts.
;-
CMPRTN: MOVB     @#RBUF,R0        ;Enter here when new character is available
                                   ;Get the new character

```


Real-Time Programs

```

        CMPB    RO,#CTRLC      ;Should we quit?
        BEQ     1$             ;Quit on ^C
        .TTYOUT                ;Display the character
        MOV     #SETPRI,RO     ;Point to EMT arg block to
        EMT     375            ;Set processor priority (block interrupts)
; . . .                        ;Time critical processing goes here
        CLR     PRILEV         ;Set priority back down
        MOV     #SETPRI,RO     ;Point to EMT arg block to
        EMT     375            ;Set processor priority (reenable interrupts)
        BR      2$
1$:      .RSUM                  ;Return to main-line code
2$:      RETURN                ;Wait for next interrupt (NOTE: Not RTI)

MAPIOP: .BYTE    5,140         ;EMT arg block to map to I/O page
LOKLOW: .BYTE    7,140         ;EMT arg block to lock job into low mem.
LOKJOB: .BYTE    13,140        ;EMT arg block to lock job in place
UNLOKJ: .BYTE    10,140        ;EMT arg block to unlock job from memory
ATTVEC: .BYTE    11,140        ;EMT arg block to attach to interrupt
        .WORD    340           ;Interrupt vector
        .WORD    CMPRTN        ;Address of completion routine
        .WORD    7             ;Real-time priority (NOT processor priority!)
RELVEC: .BYTE    12,140        ;EMT arg block to release interrupt vector
        .WORD    340           ;Interrupt vector
SETPRI: .BYTE    16,140        ;EMT arg block to set processor priority
PRILEV: .WORD    7             ;Desired priority level (modifiable)
ATTERR: .WORD    NOPRIV        ;Attach to interrupt error table
        .WORD    MAXINT
        .WORD    INUSE
        .NLIST    BEX
TYPE:   .ASCIZ    /Characters entered on serial line will be displayed here:/
BADATT: .ASCIZ    /?ATTVEC-F-Cannot attach to interrupt./<7>
NOPRIV: .ASCII    /Real-time support not specified during TSGEN or /
        .ASCIZ    /user not privileged./
MAXINT: .ASCIZ    /Maximum number of interrupts already in use./
INUSE:  .ASCIZ    /Another job already connected to interrupt./
        .END      START

```

11.12 Releasing an interrupt connection

A connection between an interrupt vector and an interrupt service routine or an interrupt completion routine remains in effect until the job exits or the following EMT is executed to release the connection.

Warning: An interrupt through a vector which has been released with this EMT or which was connected to a job that has terminated will cause a fatal system error: UEI-Interrupt occurred at unexpected location.

The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument area:

```
.BYTE    12,140
.WORD    interrupt-vector
```

where "interrupt-vector" is the address of the interrupt vector whose connection is to be released.

Example:

See the example program ATTVEC in the section on connecting a real-time interrupt to a completion routine.

11.13 Scheduling a completion routine

Real-time programs may schedule a completion routine for execution with a special EMT provided for that purpose. This is particularly valuable from within interrupt service routines which do not have access to system service calls other than this EMT and the .RSUM request. While any program with real-time (operator) privilege may issue this request, its primary intent was to provide a mechanism for access to system service calls from interrupt service routines. The form of this EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    21,140
.WORD    completion-routine
.WORD    priority
.WORD    R1-value
.WORD    0
```

where "completion-routine" is the address of the entry point of the completion routine that is being started, "priority" is the real-time priority level for the completion routine being started, and "R1-value" is a value to be placed in R1 on entry to the completion routine.

Completion routines scheduled in this manner follow the same rules as for interrupt completion routines described above, except that they are scheduled as a result of the EMT call rather than in response to an interrupt.

Real-Time Programs

Example:

See the example program INTSVC in the section on connecting interrupts to real-time jobs/ interrupt service routines.

11.14 Adapting real-time programs to TSX-Plus

The following points should be kept in mind when converting an RT-11 real-time program for use under TSX-Plus.

1. The I/O page (160000-177777) is not directly accessible to the program unless the program executes the TSX-Plus real-time EMT that maps the job's virtual region to the I/O page or is started with the /IOPAGE switch to the R[UN] command.
2. If the program's virtual region 160000 to 177777 is mapped to the I/O page, the program must use .GVAL to access offsets in the simulated RMON.
3. Since FORTRAN uses PAR 7 to map to virtual arrays, if both direct I/O page access and FORTRAN virtual arrays are required in the same program, then some PAR other than 7 must be used to map to the I/O page.
4. Real-time interrupts are connected to interrupt service routines and interrupt completion routines by use of the TSX-Plus real-time EMT for that purpose. The program should not try to connect interrupts by storing into the interrupt vector cells.
5. The .PROTECT EMT is a no-op under TSX-Plus and always returns with the carry-flag cleared.
6. Interrupt service routines and completion routines connected to real-time interrupts should exit by use of a RTS PC instruction rather than RTI.
7. Programs that require very rapid response to interrupts should use the interrupt service routine method and must lock themselves in memory.
8. Real-time interrupts must not occur unless an interrupt service routine or completion routine is connected to the interrupt. If a real-time interrupt occurs with no associated interrupt service or completion routine, a fatal TSX-Plus system error (UEI) occurs and the "argument value" displays the address of the vector of the interrupt.
9. A higher priority real-time completion routine for one job will interrupt a lower priority completion routine being executed by another job but will not interrupt a lower priority completion routine being executed by the same job.

Real-Time Programs

10. A real-time completion routine running at priority 1 or above is not time-sliced and will lock out all lower priority jobs until it completes its processing or enters a wait state.

1. The first part of the report is a general
description of the project and its objectives.
2. The second part is a detailed description of the
methodology used in the study.

12. SHARED RUN-TIME SYSTEM SUPPORT

TSX-Plus provides a facility that allows one or more shared run-time systems or data areas to be mapped into the address space of multiple TSX-Plus time-sharing jobs. There are two primary uses of this facility:

1. Memory space can be saved by having multiple jobs that use the same run-time system access a common copy rather than having to allocate space within each job for a copy.
2. Programs can communicate with each other through the use of a common shared memory region to which all of the communicating jobs have direct access.

To use this facility, information about all of the shared run-time systems must be declared when the TSX-Plus system is generated. During system initialization the shared run-time system files are opened and read into memory. These shared run-time systems remain in memory as long as the system is running and are never swapped out of memory even if there are no jobs actively using them.

The EMTs described below can be used to associate one or more shared run-time systems with a job. When such an association is made, a portion of the job's virtual memory space is mapped to allow access to part or all of one or more run-time systems.

12.1 Associating a run-time system with a job

The following EMT is used to associate a shared run-time system with a job. The form of the EMT is:

EMT 375

with R0 pointing to the following argument area:

.BYTE 0,143
.WORD name-pointer

where "name-pointer" is the address of a two-word cell containing the six character name of the shared run-time system in RAD50 form. This name corresponds to the file name which was specified for the run-time system when TSX-Plus was generated. If the name pointer value is zero, the effect of the EMT is to disassociate all shared run-time systems from the job and to re-establish normal memory mapping for the job.

If the run-time system whose name is specified with this EMT is not recognized, the carry-flag is set on return with an error code of 1.

The effect of this EMT is to associate a particular shared run-time system with the job. However, this EMT does not affect the memory mapping for the job or make the run-time system visible to the job; that is done by the EMT described below. If some other run-time system has been previously mapped into the job's region, that mapping is unaffected by this EMT. Thus it is possible to have multiple run-time systems mapped into the job's region by associating and

Shared Run-times

mapping them one at a time into different regions of the job's virtual memory space.

Example:

```
.TITLE  USERTS
.ENABL  LC
; Demonstration of TSX-Plus EMT to map to a shared run-time region
.MCALL  .PRINT,.EXIT
PAR1    = 20000                ;Base address of PAR 1 window
START:  MOV    #USERTS,RO      ;Point to EMT arg block to
      EMT      375            ;Associate a shared run-time region
      BCC      1$             ;Error?
      .PRINT   #NOSHRT        ;Can't find named shared run-time
      .EXIT
1$:     MOV    #MAPRTS,RO      ;Point to EMT arg block to
      EMT      375            ;Map to shared run-time system
2$:     CALL   @#<PAR1+200>    ;JSR to entry point in shared region
      ;which prints some data from there.
      CLR     SHRNAM          ;Undeclare any shared run-times
      CLR     <SHRNAM+2>
      MOV     #USERTS,RO      ;Point to EMT arg block to
      EMT      375            ;Dissociate all shared regions
; . . .
      .EXIT
USERTS: .BYTE   0,143          ;EMT arg block to associate
      .WORD    SHRNAM          ;Pointer to file name
SHRNAM: .RAD50  /RTCOM /      ;Shared region file name
MAPRTS: .BYTE   1,143          ;EMT arg block to map shared region
      .WORD    1               ;Map PAR 1
      .WORD    0               ;Offset into region
      .WORD    1000/64.        ;Size of region (64. byte blocks)
      .NLIST   BEX
NOSHRT: .ASCIZ  /Can't find the shared run-time system./<7>
      .END     START
```

The example program RTCOM declared in the above program was defined during TSX-Plus system generation with the RTDEF macro as follows:

```
RTDEF <SY RTCOM SAV>,RW,1
```

and the shared program itself was:

```
.TITLE  RTCOM
.ENABL  LC
; Demonstration shared run-time file
; (Position Independent Code.)
.MCALL  .PRINT
.NLIST  BEX
.PSECT  RTCOM,I               ;RTDEF in TSGEN specifies
```



```

;skip 1 block so default start address of 1000 is OK
START: .ASCII  /This data was produced by the /
       .ASCIZ  /RTCOM shared run-time region./
       . = START + 200           ;Entry point for shared code
ENTRY:: MOV    PC,R0             ;Find out where we are
       SUB     #<.-START>,R0     ;Point back to data area
       .PRINT  R0               ;Display what is there
       RETURN                      ;And go back to MAIN
       .END      ENTRY

```

12.2 Mapping a run-time system into a job's region

Once a shared run-time system has been associated with a job by use of the previous EMT, the run-time system (or a portion thereof) can be made visible to the job by mapping it into the job's virtual address region. The form of the EMT to do this is:

```
EMT      375
```

with R0 pointing to the following argument area:

```

.BYTE    1,143
.WORD    par-region
.WORD    run-time-offset
.WORD    mapped-size

```

The "par-region" parameter is a number in the range 0 to 7 that indicates the Page Address Register (PAR) that is to be used to access the run-time system. The PAR number selects the region of virtual memory in the job that will be mapped to the run-time system. The following correspondence exists between PAR numbers and virtual address regions within the job:

PAR	Address Range
0	000000 - 017777
1	020000 - 037777
2	040000 - 057777
3	060000 - 077777
4	100000 - 117777
5	120000 - 137777
6	140000 - 157777
7	160000 - 177777

The "run-time-offset" parameter specifies which portion of the run-time system is to be mapped into the PAR region. The "run-time-offset" specifies the number of the 64-byte block within the run-time system where the mapping is to begin. This makes it possible to access different sections of a run-time system at different times or through different regions.

Shared Run-times

The "mapped-size" parameter specifies the number of 64-byte blocks to be mapped. If this value is larger than can be contained within a single PAR region, multiple PAR regions are automatically mapped as necessary to contain the entire specified section of the run-time.

If an error is detected during execution of the EMT, the carry-flag is set on return with an error code of 1 to indicate that there is no run-time system associated with the job.

This EMT only affects the mapping of the PAR region specified in the argument block (and following PAR's if the size so requires). It does not affect the mapping of any other PAR regions that may previously have been mapped to a run-time system. Thus a job may have different PAR regions mapped to different sections of the same run-time system or to different run-times. Real-time programs may map PAR 7 to the I/O page and also map other PAR regions to shared run-time systems.

The memory size of a job is not affected by the use of the shared run-time control EMTs. That is, mapping a portion of a job's virtual address space to a shared run-time system neither increases nor decreases the size of memory occupied by the job. If a job's size is such that a portion of its normal memory is under a PAR region that is mapped to a run-time system, that section of its normal memory becomes inaccessible to the job as long as the run-time system mapping is in effect, but the memory contents are not lost and may be re-accessed by disassociating all run-time systems from the job.

Note that any of the PAR regions may be mapped to a run-time system including PAR 7 (160000-177777).

Example:

See the example program USERTS in the section on associating a run-time system with a job.

13. TSX-Plus PERFORMANCE MONITOR FEATURE

TSX-Plus includes a performance analysis facility that can be used to monitor the execution of a program and determine what percentage of the run time is spent at various locations within the program. During performance analysis, TSX-Plus examines the program being monitored when each clock tick occurs (50 or 60 times per second) and notes at what location in the program execution is taking place. Once the analysis is completed the TSX-Plus performance reporting program (TSXPM) can be used to produce a histogram showing the percentage of time spent at various locations during the monitored run.

There are three steps involved in performing a performance analysis on a program:

1. Use the MONITOR command to begin the analysis.
2. Run the program to be monitored.
3. Run the TSXPM program to print a histogram of the result.

13.1 Starting a performance analysis

The first step in doing a performance analysis is to use the MONITOR keyboard command to tell TSX-Plus that a performance analysis is to be done on the program that will be run next. The form of the MONITOR command is:

MONITOR base-address,top-address[,cell-size]/switches

where "base-address" is the lowest address in the region to be monitored, "top-address" is the highest address in the region to be monitored, and "cell-size" is an optional parameter that specifies the number of bytes of address in the region being monitored to be grouped together into each histogram cell. If the "cell-size" parameter is not specified, TSX-Plus calculates the cell size by dividing the number of bytes in the region being monitored (base-address to top-address) by the total number of histogram cells available (specified when TSX-Plus is generated). This gives the finest resolution possible. The only available switch is "/I" which causes I/O wait time to be included in the analysis. If this switch is not specified, only CPU execution time is included in the analysis. A link map of the program should be available to determine the addresses in the program that are appropriate to monitor.

Examples:

.MONITOR 1000,13000/I

.MONITOR 20000,40000,10

.MONITOR 2000,6000

The effect of the MONITOR command is to set up parameters within TSX-Plus which will be used to monitor the next program run. It does not actually begin the analysis, so there is no rush in running the program to be monitored. Once the MONITOR command has been issued, the program to be monitored is run by using

Performance Monitoring

the standard "RUN" or "R" commands. If a program being monitored does a .CHAIN to another program, the analysis continues and the times reported will be the composite of the programs run.

Only one user may be doing a performance analysis at a time. This is because the performance analysis histogram buffer is a common memory area that may not be in use by more than one user at a time. An analysis is in effect for a user between the time the MONITOR command is issued and the TSXPM program is run to display the results of the analysis. Running the TSXPM program terminates the performance analysis and allows other users to perform analyses. Note that space for the performance analysis data buffer must be reserved when TSX-Plus is generated.

If the program to be monitored is overlayed and the region to be monitored is in the overlay area, the analysis technique is more complex. In this situation, the EMTs described below must be used to control the performance analysis as overlay segments are run in the segment being monitored.

13.2 Displaying the results of the analysis

After the program being monitored has exited and returned control to the keyboard monitor, the TSXPM performance reporting program is used to generate a histogram of the time spent in the region being monitored. The TSXPM program is started by typing "R TSXPM"; it responds by printing an asterisk ("*"). In response to the asterisk, enter the file specification for the device/file where the histogram is to be written. An optional switch of the form "/M:nnn" may be specified following the file specification. This switch is used to specify the minimum percentage of the total run-time that a histogram cell must contain in order to be included in the display. If this switch is not specified, the default cut-off percentage is 1%.

After receiving the file specification, TSXPM prompts for a title line. Enter a line of text which will be printed as a page title in the histogram file. Press RETURN if you wish no title.

The next item of information requested by TSXPM is a set of base offset values. The base offset values are optional. Base offsets are useful in the situation where you have several modules making up a program being monitored and you want the addresses displayed on the performance analysis histogram to be relative to the base of each module. You may specify up to 10 offset values. Each offset value is specified as an offset module number (in the range 0 to 9) followed by a comma and the base address of the module (see example below). If offset values are specified, TSXPM determines in which module each cell of the histogram falls and displays the address as a module number and offset within the module. After you enter all desired module offsets, enter RETURN without a value.

After the base offset values are entered, the histogram will be produced and written to the specified device and file. After the histogram is generated, TSXPM prints the asterisk prompt again, at which point you may enter the name

of another device/file and produce the histogram again or you may type control-C to return to the keyboard monitor.

Example use of TSXPM:

```
.R TSXPM
*LP:/M:5
Title:PERFORMANCE ANALYSIS OF EIGENVALUE CALCULATION
Base offsets:
>1,1000
>2,2134
>3,5212
>
(Histogram is produced at this point)
*<CTRL-C>
```

The histogram produced by TSXPM consists of one line per histogram cell. Each line contains the following information: 1) the base module offset number (if offsets were specified); 2) the address range covered by the histogram cell (relative to the module base if base offsets were used); 3) the percentage of the total execution time spent at the address range covered by the histogram cell; 4) a line of stars presenting a graphic representation of the histogram.

13.3 Performance monitor control EMT's

For most applications the method described above can be used to do a performance analysis. However, in special cases (such as analyzing the performance of an overlaid program) it is necessary to have more explicit control over the performance analysis feature as a program is running. The following set of EMTs may be used to control a performance analysis.

13.3.1 Initializing a performance analysis.

This EMT is used to set up parameters that will control a performance analysis. It does not actually begin the analysis. The form of the EMT is:

```
EMT      375
```

with R0 pointing to the following argument block:

```
.BYTE    0,136
.WORD    base-address
.WORD    top-address
.WORD    cell-size
.WORD    flags
```

where "base-address" is the address of the base of the region to be monitored, "top-address" is the address of the top of the region to be monitored, and "cell-size" is the number of bytes to group in each histogram cell. If 0 (zero) is specified as the cell size, TSX-Plus calculates the cell size to use by dividing the number of bytes in the region being monitored (top-address

Performance Monitoring

minus base-address) by the number of cells available in the histogram data area (specified when TSX-Plus is generated). The "flags" parameter is used to control whether or not I/O wait time is to be included in the analysis. If a value of 1 is specified as the "flags" parameter, I/O wait time is included in the analysis; if a value of 0 (zero) is specified, I/O wait time is not included in the analysis.

Errors:

Code	Meaning
0	Performance analysis being done by some other user.
1	Performance analysis feature not included in TSX-Plus generation.

Example:

```
.TITLE DEMOPA
.ENABL LC

; Demonstrate use of TSX-Plus EMT's to initialize, start, stop, and
; terminate a performance analysis

.MCALL .EXIT,.PRINT,.LOOKUP,.READW,.CLOSE,.TTYOUT
.GLOBL PRT OCT          ;Display an octal word in R0

ERRBYT = 52             ;EMT error byte
SPACE  = 40             ;ASCII 'space'
STAR   = 52             ;ASCII 'asterisk'

START: MOV    #INITPA,R0 ;Point to EMT arg block to
      EMT     375        ;Initialize the performance analysis
      BCC     5$         ;No error
      MOVB    @#ERRBYT,R0 ;Which error?
      ASL     R0         ;Convert to word offset
      .PRINT  INIERR(R0) ;Print the error message
      .EXIT                    ;And depart this world of woe.

5$:   MOV     #STRTPA,R0 ;Point to EMT arg block to
      EMT     375        ;Start the performance analysis
      BCC     10$        ;No error
      .PRINT  #STRERR    ;Start error
      .EXIT                    ;and depart.

10$:

; Dummy section of code to do some I/O and computation for analysis
BEGIN: .LOOKUP #AREA,#0,#FILNAM
      MOV     #10,R1
; Disk I/O
1$:   .READW  #AREA,#0,#BUFFER,#256.,#0
      SOB     R1,1$
      .CLOSE  #0
```


Performance Monitoring

```

; Terminal I/O
    .PRINT #BUFFER
; Compute bound
    MOV     #123456,R3
2$:    MOV     #12345,R0
        CLR     R1
        DIV     #345,R0
        SOB     R3,2$
ENDB:
NCELLS = <<ENDB-BEGIN>/2>                ;Number of cells in histogram

    MOV     #STOBLK,R0                    ;Point to EMT arg block to
    EMT     375                          ;Stop the performance analysis
    BCC     15$                          ;Check for error return
    .PRINT  #STOERR                      ;Only one error - PA not initialized
    .EXIT                                     ;and leave.

15$:    MOV     #HALBLK,R0                ;Put HALTPA block address in R0
    EMT     375                          ;Terminate the performance analysis
    BCC     20$                          ;Check for errors
    MOVB    @#ERRBYT,R0                  ;Get error type
    ASL     R0                          ;Convert to word offset
    .PRINT  HLERR(R0)                   ;Print out proper error message
    .EXIT                                     ;And depart.

20$:    TST     HALFLG                    ;Check HALTPA return flag
    BPL     25$                          ;No cell overflow?
    .PRINT  #OVRWRN                      ;Issue overflow warning

25$:    BIT     #1,HALFLG                 ;I/O time included?
    BEQ     30$                          ;Skip message if not
    .PRINT  #IOWNOT                      ;Print I/O wait time included msg

; Print a histogram of the performance analysis
30$:    CLR     R3                        ;Set histogram cell counter
35$:    CMP     HSTTBL(R3),#64.           ;Normalize the table for 64. longest
        BHI     40$                      ;Too many counts?
        ADD     #2,R3                    ;Point to next cell
        CMP     R3,#<2*NCELLS>           ;End of table?
        BLE     35$                      ;No, repeat
        BR      50$                      ;All cells less than 64. counts now
40$:    CLR     R3                        ;Re-init. histogram cell counter
45$:    CLC                                     ;Divide each cell count by 2
        ROR     HSTTBL(R3)
        ADD     #2,R3                    ;Point to next cell
        CMP     R3,#<2*NCELLS>           ;End of table?
        BLE     45$                      ;No, repeat
        BR      50$                      ;Go check all cells again
50$:    .PRINT  #HSTMSG                   ;Caption histogram
        MOV     #BEGIN,R2                ;Set first analyzed address
        CLR     R3                        ;Init. cell counter
55$:    MOV     R2,R0                      ;Get ready to print it out

```


Performance Monitoring

```

CALL      PRTOUT      ;Display analyzed address
.TTYOUT   #SPACE      ;For formatting
MOV       HSTTBL(R3),R1 ;Get address use counter
BEQ       65$         ;No stars on 0 count
60$:      .TTYOUT   #STAR ;Print a '*' for each count
SOB       R1,60$
65$:      .PRINT    #CRLF ;Go to next line
CMP       (R2)+,(R3)+  ;Point to next address and count
CMP       R3,#<2*NCELLS> ;End of histogram?
BLE       55$         ;No, display next cell
.EXIT     ;Else done.

INITPA:   .BYTE      0,136 ;EMT arg block for perform. analysis
          .WORD      BEGIN ;Start analysis address
          .WORD      ENDB-2 ;End analysis address
          .WORD      2      ;Count 1 address in each cell
          .WORD      1      ;Include IO wait time.
STRTPA:   .BYTE      1,136 ;Start Performance Analysis EMT block
STOBLK:   .BYTE      2,136 ;EMT arg block to stop perf. analysis
HALBLK:   .BYTE      3,136 ;EMT arg block to release analysis
          .WORD      PRMBUF ;PA parameter buffer address
          .WORD      HSTTBL ;Histogram Table buffer address
          .WORD      <HSTEND-HSTTBL> ;Histogram Table buffer length
PRMBUF:   .BLKW      3      ;PA four word parameter buffer
HALFLG:   .WORD      0      ;PA return flags
HSTTBL:   .BLKW      <2*NCELLS>+2 ;Histogram table
HSTEND:

AREA:     .BLKW      10     ;EMT arg block area
BUFFER:   .BLKW      256.   ;Data input buffer
          .WORD      0      ;Make sure buffer is ASCIZ
FILNAM:   .RAD50     /DK DEMOPAMAC/ ;Read this file
INIERR:   .WORD      INPRG  ;Initialize PA error table
          .WORD      NOGEN
HLTERR:   .WORD      NOPA   ;HALTPA Error message table
          .WORD      TOSMAL
          .NLIST BEX
INPRG:    .ASCIZ     /?INITPA-F-Performance analysis being done by another user./
NOGEN:    .ASCIZ     /?INITPA-F-Performance analysis feature not gennded./
NOPA:     .ASCIZ     /?HALTPA-F-This job is not doing a performance analysis./
TOSMAL:   .ASCIZ     /?HALTPA-F-Area provided for histogram table too small./
STRERR:   .ASCIZ     /?INITPA-F-Performance analysis not initialized yet./
STOERR:   .ASCIZ     /?STOPPA-F-Performance Analysis has not been initialized./
OVRWRN:   .ASCIZ     /?HALTPA-W-Some histogram cell(s) overflowed during analysis./
IOWNOT:   .ASCIZ     | HALTPA-I-I/O wait time included in performance analysis.|
OKDONE:   .ASCIZ     / STOPPA-S-Performance Analysis stopped./
HSTMSG:   .ASCIZ     /Address      Frequency/
CRLF:     .ASCIZ     <15><12><200>
          .END       START

```


13.3.2 Starting a performance analysis.

This EMT is used to begin the actual collection of performance analysis data. The previous EMT must have been executed to set up parameters about the performance analysis before this EMT is called. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 1,136

The carry-flag will be set on return from this EMT if performance analysis has not been initialized yet.

Example:

See the example program DEMOPA in the section on initializing a performance analysis.

13.3.3 Stopping a performance analysis.

The following EMT can be used to suspend the data collection for a performance analysis. The data collection can be restarted by using the start-analysis EMT described above. This EMT could, for example, be used to suspend the analysis when an overlay module is loaded that is not to be monitored. The start-analysis EMT would then be used to re-enable the data collection when the overlay of interest is re-loaded. The form of this EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 2,136

The carry flag will be set on return from this EMT if performance analysis has not been initialized yet.

Example:

See the example program DEMOPA in the section on initializing a performance analysis.

13.3.4 Terminating a performance analysis.

This EMT is used to conclude a performance analysis. It has the effect of returning into a user supplied buffer the results of the analysis and releasing the performance analysis feature for other users. The form of this EMT is:

Performance Monitoring

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 3,136
.WORD parameter-buffer
.WORD histogram-buffer
.WORD buffer-size
```

where "parameter-buffer" is the address of a 4 word buffer into which will be stored some parameter values describing the analysis that was being performed, "histogram-buffer" is the address of the buffer that will receive the histogram count values, and "buffer-size" is the size (in bytes) of the histogram buffer area.

The values returned in the parameter buffer consist of the following 4 words: 1) base address of the monitored region; 2) top address of the monitored region; 3) number of bytes per histogram cell; 4) control and status flags. The control and status flags are a set of bits that provide the following information:

Flag	Meaning
1	I/O wait time was included in the analysis.
100000	Some histogram cell overflowed during the analysis.

The data returned in the histogram buffer consists of a vector of 16-bit binary values, one value for each cell in the histogram. The first value corresponds to the histogram cell that starts with the base address of the region that was being monitored.

Errors:

Code	Meaning
0	This job is not doing a performance analysis.
1	Area provided for histogram count vector is too small.

Example:

See the example program DEMOPA in the section on initializing a performance analysis.

13.3.2 Starting a performance analysis.

This EMT is used to begin the actual collection of performance analysis data. The previous EMT must have been executed to set up parameters about the performance analysis before this EMT is called. The form of the EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 1,136

The carry-flag will be set on return from this EMT if performance analysis has not been initialized yet.

Example:

See the example program DEMOPA in the section on initializing a performance analysis.

13.3.3 Stopping a performance analysis.

The following EMT can be used to suspend the data collection for a performance analysis. The data collection can be restarted by using the start-analysis EMT described above. This EMT could, for example, be used to suspend the analysis when an overlay module is loaded that is not to be monitored. The start-analysis EMT would then be used to re-enable the data collection when the overlay of interest is re-loaded. The form of this EMT is:

EMT 375

with R0 pointing to the following argument block:

.BYTE 2,136

The carry flag will be set on return from this EMT if performance analysis has not been initialized yet.

Example:

See the example program DEMOPA in the section on initializing a performance analysis.

13.3.4 Terminating a performance analysis.

This EMT is used to conclude a performance analysis. It has the effect of returning into a user supplied buffer the results of the analysis and releasing the performance analysis feature for other users. The form of this EMT is:

Performance Monitoring

EMT 375

with R0 pointing to the following argument block:

```
.BYTE 3,136
.WORD parameter-buffer
.WORD histogram-buffer
.WORD buffer-size
```

where "parameter-buffer" is the address of a 4 word buffer into which will be stored some parameter values describing the analysis that was being performed, "histogram-buffer" is the address of the buffer that will receive the histogram count values, and "buffer-size" is the size (in bytes) of the histogram buffer area.

The values returned in the parameter buffer consist of the following 4 words: 1) base address of the monitored region; 2) top address of the monitored region; 3) number of bytes per histogram cell; 4) control and status flags. The control and status flags are a set of bits that provide the following information:

Flag	Meaning
-----	-----
1	I/O wait time was included in the analysis.
100000	Some histogram cell overflowed during the analysis.

The data returned in the histogram buffer consists of a vector of 16-bit binary values, one value for each cell in the histogram. The first value corresponds to the histogram cell that starts with the base address of the region that was being monitored.

Errors:

Code	Meaning
----	-----
0	This job is not doing a performance analysis.
1	Area provided for histogram count vector is too small.

Example:

See the example program DEMOPA in the section on initializing a performance analysis.

14. TSX-Plus RESTRICTIONS

14.1 System service call (EMT) differences between RT-11 and TSX-Plus

The following list describes the differences in the way TSX-Plus implements some RT-11 system service calls (EMTs). If an EMT is not listed, it provides the same functions as described in the RT-11 Programmer's Reference Manual.

- .ABTIO Action depends on setting of IOABT system parameter. If IOABT is set to 1 then .ABTIO operates the same as RT-11 and calls handler abort entry points. If IOABT is set to 0 then .ABTIO does not call handler entry points, but instead does a .WAIT on all channels. The default method of handling I/O abort requests is chosen during TSX-Plus system generation with the IOABT parameter. The I/O abort handling method may also be changed while TSX-Plus is running with the SET IO [NO]ABORT keyboard command.
- .CDFN User jobs may not define more than 16 channels. If a .CDFN EMT is done, all channels are purged if a .CHAIN is done. If .CDFN is not done, channels are not purged across a chain.
- .CHCOPY Not implemented.
- .CNTXSW Not implemented.
- .DEVICE Operator privilege is required to use this EMT.
- .FETCH Returns in R0 the address specified for the handler load area. All TSX-Plus device handlers are resident, hence the .FETCH EMT performs no operation. If the device handler specified was not loaded during TSX-Plus initialization, then an error code of 0 is returned.
- .FORK May be used in device handlers but not in user jobs.
- .GTJB The job number returned in word 1 of the result area is two times the TSX-Plus line number. That is, the first line specified in the system generation will be number 2, the second 4, etc. Words 8 through 12 of the result area are not altered by this EMT.
- .HRESET Action depends on setting of IOABT system generation parameter. If IOABT is set to 1 then .HRESET calls handler abort entry points. If IOABT is set to 0 then .HRESET does not call handler entry points but instead does a .WAIT on all channels. Messages queued on named message channels are not canceled. Otherwise, this EMT operates the same as under RT-11.
- .INTEN May be used in device handlers but not in user jobs.
- .LOCK The TSX-Plus file management module (USR) is always "resident" and the .LOCK EMT is ignored.
- .MTxxxx Multi-terminal control EMT's (.MTIN, .MTOUT, .MTPRNT, etc.) are not supported.

Restrictions

- .MTPS The processor priority level may not be changed from user mode, hence this macro performs no operation. TSX-Plus provides a special real-time EMT to set the processor priority.
- .MWAIT Not supported. See the chapter that describes inter-job message communication.
- .PEEK Non-privileged jobs may use .PEEK to access cells within the simulated RMON (addresses 160000 to 160626) although the .GVAL EMT is a recommended alternative. Jobs with operator privilege may use .PEEK to access the I/O page or low memory cells in kernel space. References to addresses in the virtual address range of the simulated RMON (160000 to 160626) are always directed to RMON rather than the I/O page.
- .POKE Non-privileged jobs may use .POKE to access cells within the simulated RMON (addresses 160000 to 160626) although the .PVAL EMT is a recommended alternative. Jobs with operator privilege may use .POKE to access the I/O page or low memory cells in kernel space. References to addresses in the virtual address range of the simulated RMON (160000 to 160626) are always directed to RMON rather than the I/O page.
- .PROTECT Not supported. See the chapter on real-time programming for information about how to connect an interrupt to a TSX-Plus job.
- .QSET TSX-Plus uses an internal pool of I/O queue elements for all jobs hence it is not necessary to define additional I/O queue elements in order to perform overlapped I/O. The .QSET EMT is ignored.
- .RCVD Not supported. See the chapter that describes inter-job message communication.
- .RELEAS This EMT is ignored. Refer to .FETCH.
- .SETTOP Returns the job limit in R0 but does not actually expand or contract the allocated job region. The job region allocation can be changed by use of the MEMORY keyboard command or the TSX-Plus specific EMT for this purpose.
- .SDAT Not supported. See the chapter that describes inter-job message communication.
- .SFPA This EMT functions the same as RT-11. It must be used if a job is going to use the floating-point unit.
- .SDTTM Operator privilege is required to use this EMT.
- .SRESET Messages queued on named message channels are not canceled. Otherwise, this EMT operates the same as under RT-11.

- .SYNCH May be used in handlers but not in user jobs. When used in a handler, the number of the TSX-Plus job that is being synchronized with must be stored in word 2 of the SYNCH block.
- .TLOCK This EMT is ignored.
- .TTINR Only honors bit 6 in the Job Status Word (TCBIT\$) if a SET TT NOWAIT command has been issued, or the running program has send the "U" program controlled terminal option to the system, or the program was R[UN] with the /SINGLECHAR switch.
- .UNLOCK This EMT is ignored.
- .UNPROT Not supported. See the chapter on real-time programming for information about how to connect an interrupt to a TSX-Plus job.

A full list of RT-11 compatible and TSX-Plus specific EMTs is included in Appendix D. The list indicates the level of support TSX-Plus provides for each RT-11 compatible EMT.

14.2 Programs Not Supported by TSX-Plus

Most programs which run under RT-11 will run under TSX-Plus without change.

The TSX-Plus program debugging facility (see Appendix I) should be used rather than ODT.

The FORMAT program may not be used under TSX-Plus.

The BATCH RT-11 facility is not supported by TSX-Plus.

The logical subset disk feature is provided internally to TSX-Plus and therefore does not use the LD pseudo-device handler.

The single line editor feature is provided as an optional system overlay region with TSX-Plus and does not use the SL pseudo-device handler.

The VM handler supplied with TSX-Plus was written specially for use with TSX-Plus and is not the same as the DEC VM handler.

The QUEUE package (QUEUE and QUEMAN) is not supported by TSX-Plus.

The RESORC utility is not supported.

14.3 Special program suggestions

Certain programs use system resources in ways which may not seem obvious or cause behaviour which is not expected. Some of those features are described in this section to facilitate their use with TSX-Plus.

Restrictions

14.3.1 DIBOL:

When DIBOL programs are run under TSX-Plus they must use SUD rather than TSD. Several components of the DIBOL system also use I/O channel numbers above 15 (decimal). TSX-Plus normally only allows jobs to access channels 0 through 15 (decimal). This can result in the error ?INI-E3 or ?INI-E4 when using DIBOL or its utilities. This can be resolved either by patching SUD or the utility to use lower channel numbers or by regenerating TSX-Plus to permit use of higher channel numbers. Set the TSGEN parameter NUCHN to 20., reassemble TSGEN, and relink TSX-Plus.

Keep in mind that record locking does not automatically occur when DIBOL is used with TSX-Plus and unless the appropriate TSX-Plus record locking calls are made there is a danger of file corruption. Appendix B describes subroutines which may be linked into DIBOL programs to provide access to TSX-Plus record locking and inter-program messages. Do not use the DIBOL SEND statement, use an equivalent XCALL MSEND instead.

14.3.2 DU device handler:

The DU device handler is used to access several types of devices, including large disks (like the RA-80). Because the RT-11 directory structure cannot support disk block numbers larger than 65535, this restricts the size of disks which may be used with RT-11 and TSX-Plus. Some devices using the MSCP protocol through the DU handler may be partitioned into several smaller devices to alleviate the block number problem. In order to use such devices with TSX-Plus, first boot RT11XM, partition DU as desired using the SET command, and finally copy the handler (now with the appropriate options set) to use with TSX-Plus. A typical sequence of commands might look like:

```
BOOT SY:RT11XM
SET DU0 UNIT=0,PORT=0,PART=0
SET DU1 UNIT=0,PORT=0,PART=1
SET DU2 UNIT=0,PORT=0,PART=2
SET DU3 UNIT=0,PORT=0,PART=3
SET DU4 UNIT=1
BOOT SY:RT11SJ
COPY/SYS SY:DUX.SYS SY:DU.TSX
R TSX
```

14.3.3 IND .ASKx timeouts:

When using the IND control file processor, the .ASKx directives provide an optional timeout parameter. This parameter causes IND to proceed when no terminal response is obtained before the designated timeout period. When using .ASKx timeouts under TSX-Plus, it is necessary to SET TT NOWAIT to permit IND to proceed when the timeout period expires.

14.3.4 FORTRAN virtual arrays:

When the object-time system for FORTRAN-IV is generated, three options are available relating to use of virtual arrays: NOVIR, VIRP, or VIRNP. If virtual arrays are to be used at all, either VIRP or VIRNP must be selected. Under RT11SJ, the VIRNP option is commonly selected. This causes FORTRAN

virtual array handling to directly manipulate the memory management registers in the I/O page. Clearly, in a multi-user environment, like TSX-Plus, this is not advisable. The VIRP option is used with RT11XM and must also be selected for use with TSX-Plus. TSX-Plus must also be generated with PLAS support; see your system manager. If the wrong type of virtual array support is selected, various types of FORTRAN errors may occur, including virtual array initialization failures.

FORTRAN-IV uses PAR 7 to map to virtual arrays. (Page Address Register 7 maps virtual addresses 160000 to 177777.) TSX-Plus normally maps a job's PAR 7 to a simulated RMON so that older programs which require direct access to fixed offsets in RMON may operate without modification. When a program requires direct access to the I/O page to manipulate device registers, that is commonly done by mapping PAR 7 to the I/O page. This can either be done with the /IOPAGE switch to the R[UN] command or with a TSX-Plus EMT. Mapping PAR 7 to the I/O page only conflicts slightly with direct RMON access since the parameters available there may also (and should be) obtained with the .GVAL request (SYSLIB ISPY function). However, because use of FORTRAN virtual arrays also affects PAR 7 mapping, mapping PAR 7 to the I/O page causes more severe problems when programs must have both virtual arrays and direct I/O page access. The solution to this is to map some other part of the job's virtual address space to the I/O page. TSX-Plus provides an EMT to map any part of a job's address space to any physical region, including the I/O page. See Chapter 11 for information on mapping to a physical region to resolve the virtual array vs. I/O page conflict.

14.3.5 MicroPower/Pascal:

Various program components of MicroPower/Pascal (TM of Digital Equipment Corporation) should be run as virtual programs. That is, they should run without direct RMON access. Under TSX-Plus, this permits them to obtain a virtual job address space of a full 64Kb. TSX-Plus must also be generated to allow 64Kb jobs. To define the Pascal compiler and the PASDBG and MIB programs as virtual jobs, use the SETSIZ program (see Appendix A) to allow a 64Kb memory partition for them. For example:

```
.R SETSIZ
*SY: PASDBG/T:64.
**C
```

14.3.6 Overlaid programs:

TSX-Plus permits programs to open I/O channels 0 through 17 (octal). However, the overlay handler uses channel 17 (octal). Therefore, overlaid programs should not use I/O channel 17 (octal).

14.3.7 VTCOM:

The communication program VTCOM supplied with RT-11 is used for connecting a terminal on one computer system as a virtual terminal on another computer system and for file transfers between two systems. The XM version of VTCOM (VTCOM.SAV, not VTCOM.REL) must be used with TSX-Plus, and this requires that TSX-Plus be generated to include PLAS (XM) support. Under RT-11, VTCOM uses

Restrictions

the XL device handler for the low-level interface protocol. Under TSX-Plus, the CL facility replaces the XL handler and provides more flexibility in use of communication lines. CL units may either be designated as dedicated CL lines and used only as I/O devices, or may be used to "take over" inactive time-sharing lines. Either type of CL unit may be used with VTCOM. However, whichever CL unit is attached to the remote computer (possibly through a modem) must be assigned the device name XL (XC on the Pro 350) so that VTCOM can access it. The designated CL unit should also be set NOLFOUT. To avoid interference between two persons attempting to access the same CL unit at the same time (and thereby causing great confusion) the designated CL unit should also be allocated for exclusive use. The following example command file indicates how a time-sharing line may be "taken over" and used with the VTCOM program.

```
SET CLO LINE=6      !Take over time-sharing line
ASSIGN CLO XL        !Assign for VTCOM
ALLOCATE XL          !Prevent multi-user collisions
SET XL NOLFOUT       !Inhibit line feed transmission
R VTCOM              !Part of RT-11 5.01 distribution
!Communicate with remote system
DEALLOCATE XL        !Let others use CLO
DEASS XL              !Clean up assignment
SET CLO LINE=0       !Re-enable time-sharing line
```


Appendix A -- SETSIZ PROGRAM

The SETSIZ program can be used to store into a SAV file information about how much memory space should be allocated for the program when it is executed. SETSIZ can also be used to set the "virtual job" flag in the job status word (JSW) for a program.

There are three ways that the amount of memory allocated to a job can be controlled:

1. The TSX-Plus EMT with function code 141 (described in Chapter 7) may be used by a running program to dynamically set the job's size.
2. If a size is specified in a SAV file (by use of the SETSIZ program) the specified amount of memory is allocated for the program when it is started.
3. If no size is specified in the SAV file, the size specified by the last MEMORY keyboard command is used.

Note that the .SETTOP EMT does not alter the amount of memory space allocated to a job but can be used by a running program to determine the amount of memory allocated.

The effect of the SETSIZ program is to store into location 56 of block 0 of the SAV file the number of K-words of memory to allocate for the program when it is run (the LINK "/K:n" switch can also be used to do this). This value has no effect when the SAV file is run under RT-11 but causes TSX-Plus to allocate the specified amount of memory when starting the program.

If a size value is specified in a SAV file, it takes precedence over the size specified by the last MEMORY keyboard command. The TSX-Plus EMT with function code 141 may still be used to dynamically alter the memory allocation while the program is running.

Most programs allocate memory in two ways: 1) a static region that includes the program code and data areas of fixed size; 2) a dynamic region that is allocated above the static region -- usually the .SETTOP EMT is used to determine how much dynamic space is available to the program. The size of the static region for a program is fixed at link time. If the program is overlaid the static region includes space for the largest overlay segment as well as the program root. Location 50 in block 0 of the SAV file is set by the linker to contain the address of the highest word in the static region of the program.

The amount of memory space to allocate for a SAV file can be specified to the SETSIZ program in either of two ways: 1) as the total amount of memory for the program which includes space for the static plus dynamic regions; or 2) as the amount of memory for the dynamic region only, in which case SETSIZ automatically adds the size of the static region.

SETSIZ

A.1 Running the SETSIZ program

The SETSIZ program is started by use of the command

.R SETSIZ

it responds by printing an asterisk to prompt for a command line. The form of the command line is:

*filespec/switch:value

Where "filespec" is a file specification of the standard form dev:name.ext with the default device being "DK" and the default extension being "SAV".

If a file specification is entered without a switch, the effect is to cause SETSIZ to display information about the size of the SAV file; the SAV file is not altered.

SETSIZ also displays the following status message if the SAV file is flagged as being a virtual image.

Virtual-image flag is set

The virtual message is displayed if either of the following two conditions exist for the SAV file:

1. Bit 10 (mask 2000) is set in the job status word (location 44) of the SAV file.
2. Location 0 of the SAV file contains the RAD50 value for "VIR".

These are the same two conditions that cause TSX-Plus to recognize the SAV file as being a virtual image when it is started.

Examples:

.R SETSIZ

*TSTPRG

Base size of program is 22Kb

Size of allocation space is 28Kb

*SY:PIP

Base size of program is 10Kb

Size of allocation space is 22Kb

*PROG1

Base size of program is 31Kb

No allocation size specified in SAV file

Virtual-image flag is set

A.2 Setting total allocation for a SAV file

The "/T" switch is used to specify the total amount of memory space to be allocated for a program when it is run. The form of the /T switch is "/T:value." where "value" is the number of K-bytes of memory to allocate. Note that a decimal point must be specified with the value if it is entered as a decimal value.

If the "/T" switch is used without a value, the effect is to clear the TSX-Plus size allocation information in the SAV file.

Examples:

```
.R SETSIZ
*SY:PIP/T:18.
*TSTPRG/T:32.
*PROG1/T
```

A.3 Setting amount of dynamic memory space

The "/D" switch is used to specify the amount of dynamic memory space to be reserved for a program. The SETSIZ program calculates the total amount of space to allocate for the program by adding the static size (stored in location 50 of the SAV file by LINK) to the specified dynamic size. The form of the /D switch is "/D:value." where "value" is the number of K-bytes of dynamic memory space to reserve. If a program does not use any dynamic memory space, the /D switch may be used without an argument value to cause the total memory space allocation to be set equal to the static size of the program. FORTRAN programs use dynamic space for I/O buffers and the exact amount required depends on the number of I/O channels used. However, 4Kb of dynamic memory space seems to be adequate for most FORTRAN programs.

Examples:

```
.R SETSIZ
*SY:PIP/D:11.
*TSTPRG/D:4.
*PROG1/D
```

A.4 Setting virtual-image flag in SAV file

The "/V" switch is used to cause SETSIZ to set the virtual-image flag in the SAV file. This flag is bit 10 (mask 2000) in the job status word (location 44) of the SAV file.

Setting this flag indicates that the program will not directly access RMON, although it may do so indirectly by use of the .GVAL and .PVAL EMT's. The significance with regard to TSX-Plus is that it allows the program to use more than 56Kb (if that much memory is also allowed by use of a MEMORY command). The virtual-image flag should not be set unless you are sure the job does not need direct access to the RMON area.

1944

The first part of the report deals with the general situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The second part of the report deals with the economic situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The third part of the report deals with the political situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The fourth part of the report deals with the social situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The fifth part of the report deals with the cultural situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The sixth part of the report deals with the scientific situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The seventh part of the report deals with the religious situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The eighth part of the report deals with the legal situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

The ninth part of the report deals with the administrative situation of the country and the progress of the war. It is followed by a detailed account of the military operations in the various theaters of war.

Appendix B -- DIBOL TSX-Plus SUPPORT SUBROUTINES

A set of subroutines is provided with TSX-Plus to perform DIBOL record locking and message transmission functions. DIBOL ISAM files are not supported by TSX-Plus. These subroutines may not be used with DBL, which provides most of their functionality separately. Note that if these TSX-Plus features are to be used, they must be enabled when the TSX-Plus system is generated.

B.1 Record locking subroutines

The record locking subroutines coordinate access to a common file being shared and updated by several TSX-Plus users. The five subroutines parallel the operation of the DIBOL statements: OPEN, CLOSE, READ, WRITE and UNLOCK. The normal DIBOL I/O statements cannot be used to perform record locking under TSX-Plus.

Opening the file

The first subroutine is used to open a shared file in update mode. The form of the call is:

```
XCALL FOPEN(chan,devlbl,errflg)
```

where

chan = A decimal expression that evaluates to a number in the range 1-15. This is the channel number used in associated calls to FREAD, FWRIT, FUNLK and FCLOS subroutines.

devlbl = The name of an alphanumeric literal, field or record that contains the file specification in the general form: dev:filnam.ext

The file size must not be specified with the file name. An optional "/W" switch may be appended to the file name to cause the "WAITING FOR dev:file" message to be printed.

errflg = A numeric variable capable of holding at least two digits into which is stored an indication of the result of the FOPEN call. The following values are returned:

Value	Meaning
-----	-----
0	No error. File is open and ready for access.
17	File name specification is invalid.
18	File does not exist or channel is already open.
72	Too many channels are open to shared files.
	Re-gen TSX-Plus and increase the value of MAXSFC parameter.
73	Too many shared files are open.
	Re-gen TSX-Plus and increase the value of MAXSF parameter.

The FOPEN subroutine should only be used to open files that will be updated by several users. The normal DIBOL OPEN READ/WRITE sequence should be used for other files. Several files may be opened for update by calling FOPEN with different channel numbers. The ONERROR DIBOL statement does not apply to these record locking subroutines. Instead the "errflg" argument is used to indicate the outcome of the operation.

DIBOL Subroutines

Locking and reading a record

The FREAD subroutine is used to lock and read a record. The form of the call is:

XCALL FREAD(chan,record,rec #,'T' or 'W',errflg)

where

chan = Decimal expression in the range 1-15 that identifies a channel previously opened by FOPEN.

record = Name of the record or alphanumeric field in which the record read is to be placed.

rec # = Decimal expression that specifies the sequence number of the record to be read. This value must be between 1 and the total number of records in the file.

'T'/'W' = If 'T' is specified as the fourth parameter, FREAD will return a value of 40 in errflg if the requested record is locked by some other user. If 'W' is specified, FREAD will wait until the record is unlocked by all other users and will never return the record-locked error code.

errflg = Decimal variable into which is stored one of the following values:

Value	Meaning
-----	-----
0	No error. Record has been locked and read.
1	End-of-file record has been read.
22	I/O error occurred on read or channel is not open.
28	Invalid record number (possibly beyond end of file).
40	Record locked by another user. (Only returned if 'T' is specified as fourth argument.)
71	Channel was not opened by calling FOPEN.
72	Request to lock too many blocks in file. Re-gen TSX-Plus and increase value of MXLBLK parameter.

The FREAD subroutine functions like the DIBOL READ statement. However, whereas the DIBOL READ statement always returns an error code (40) if the requested record is locked, FREAD offers the user a choice: If 'T' is specified as the fourth argument to FREAD, a code of 40 will be returned in errflg if the record is already locked. If 'W' is specified as the fourth argument and the record is locked, FREAD does not return an error code, but rather waits until the requested record is unlocked. It is much more efficient to wait for a locked record by using the 'W' option rather than re-executing the FREAD with the 'T' option. It may be desirable to perform the first FREAD using the 'T' option. If the record is locked a "WAITING FOR RECORD..." message can be displayed on the user's console and another FREAD can be issued with the 'W' option to wait for the record. On return from this FREAD the "WAITING" message can be erased.

Note that although record locking is requested on a record-by-record basis, the actual locking is done on a block-within-file basis. (A block contains 512 characters). The result of this is that a record is locked if any record contained in the same block(s) as the desired record is locked.

Once a record is locked and read using FREAD, the record remains locked until the program performs one of the following operations:

1. Issues an FWRIT to the channel from which the record was read.
2. Issues another FREAD to the channel.
3. Issues an FUNLK to the channel.
4. Issues an FCLOS to the channel.
5. Terminates execution by use of the STOP statement or because of an error.

The same set of rules that apply to the DIBOL READ statement apply to FREAD.

Writing a record

The FWRIT subroutine is called to write a record to a shared file. The form of the call is:

```
XCALL FWRIT(chan,record,rec #,errflg)
```

where

chan = Channel number associated with the file.

record = Name of the record or alphanumeric field that contains the record to be written.

rec # = Decimal expression that specifies the sequence number of the record to be written.

errflg = Decimal variable into which is stored one of the following values.

Value	Meaning
-----	-----

0 No error.

22 I/O error occurred during write or channel is not open.

28 Bad record number specified.

The FWRIT subroutine writes the indicated record to the file then unlocks any blocks that were locked by the program. FWRIT appends a <CR><LF> to the end of the written record as does the DIBOL WRITE statement. The rules for the DIBOL WRITE statement also apply to FWRIT.

DIBOL Subroutines

Unlocking records

The FUNLK subroutine is used to unlock records that were locked by calling FREAD. The form of the call is:

```
XCALL FUNLK(chan)
```

chan = Channel number.

Closing a shared file

The FCLOS subroutine is called to close a channel that was previously opened to a shared file by calling FOPEN. The form of the call is:

```
XCALL FCLOS(chan)
```

chan = Channel number.

FCLOS unlocks any locked records and closes the file. Other users accessing the file are unaffected. After calling FCLOS, the channel may be reopened to some other file.

Record Locking Example

In the following example a program performs the following functions:

1. Opens a shared file named "INV.DAT" on channel 2.
2. Reads a record whose record number is stored in RECN into the field named ITEM and waits if the record is locked by another user.
3. Updates the information in the record.
4. Rewrites the record to the same position in the file.
5. Closes the shared file.

```
XCALL FOPEN(2,'INV.DAT',ERRFL)
XCALL FREAD(2,ITEM,RECN,'W',ERRFL)
; <update record>
XCALL FWRIT(2,ITEM,RECN,ERRFL)
XCALL FCLOS(2)
```

Modifying programs for TSX-Plus

It is a straightforward process to modify DIBOL programs to use the TSX-Plus record locking subroutines. OPEN, CLOSE, READ, WRITE, and UNLOCK statements that apply to shared files must be replaced by the appropriate subroutine calls. Error conditions must be tested by IF statements following the subroutine calls rather than by using the ONERROR statement.

B.2 Message communication subroutines

Three subroutines are included in the DIBOL support package to allow programs to transfer messages to each other. When running under TSX-Plus these subroutines must be used instead of the DIBOL SEND and RECV statements.

Message Channels

Messages are transferred to and from programs by using TSX-Plus "Message Channels". A message channel accepts a message from a sending program, stores the message in a queue associated with the channel and delivers the message to a receiving program that requests a message from the channel. Message channels are totally separate from I/O channels.

Each active message channel has associated with it a one to six character name that is used by the sending and receiving programs to identify the channel. The total number of message channels is defined when TSX-Plus is generated. The names associated with the channels are defined dynamically by the running programs. A message channel is said to be "active" if any messages are being held in the queue associated with the channel or if any program is waiting for a message from the channel. When message channels become inactive they are returned to a free pool and may be reused by another program.

The DIBOL SEND command directs a message to a program by using the name of the receiving program. Under TSX-Plus, a sending program transmits a message using an arbitrary channel name. Any program may receive the message by using the same channel name when it requests a message.

Sending a Message

The MSEND subroutine is called to queue a message on a named channel. If other messages are already pending on the channel the new message is added to the end of the list of waiting messages. The form of the call is:

```
XCALL MSEND(chan,message,errflg)
```

where

chan = alphanumeric literal or variable that contains the channel name (1 to 6 characters).

message = alphanumeric or decimal literal, field or record that contains the message to be sent.

errflg = Decimal variable into which will be stored one of the following values:

Value	Meaning
-----	-----
0	No error. Message has been sent.
1	All message channels are busy. (Re-gen TSX-Plus and increase the value of MAXMC parameter).
2	Maximum allowed number of messages are being held in message queues. (Re-gen TSX-Plus and increase the value of MAXMSG parameter).

Note that the maximum message length that may be transferred is defined during system generation by the MSCHRS parameter. If a message longer than this is sent, only the first part of the message will be delivered.

DIBOL Subroutines

Checking for Pending Messages

The MSGCK subroutine may be called to determine if any messages are pending on a named channel. The form of the call is:

XCALL MSGCK(chan,message,errflg)

where

chan = alphanumeric literal or variable that contains the name of the channel (1 to 6 characters).

message = alphanumeric or decimal field or record where the received message is to be placed.

errflg = decimal variable into which will be stored one of the following values:

Value	Meaning
-----	-----
0	No error. A message has been received.
3	No message was queued on the named channel.

If a received message is shorter than the receiving message field the remainder of the field is filled with blanks. If the message is longer than the field, only the first part of the message is received.

Waiting for a Message

The MSGWT subroutine is used by a receiving program to suspend its execution until a message is available on a named channel. It is much more efficient for a program to wait for a message by calling MSGWT rather than repeatedly calling MSGCK. The form of the call is:

XCALL MSGWT(chan,message,errflg)

where the arguments have the same meaning as for MSGCK, and the following values may be returned in errflg.

Value	Meaning
-----	-----
0	No error. A message has been received.
1	All message channels are busy.

Message Examples

In the following example a program sends a message to another program by using a message channel named "SORT" and then waits for a reply to come back through a message channel named "REPLY".

```

XCALL MSEND('SORT','DK:PAYROL.DAT',ERRFL)
IF(ERRFL.NE.0)GO TO ERROR
XCALL MSGWT('REPLY',MSGBF,ERRFL)
IF(ERRFL.NE.0)GO TO ERROR

```

B.3 Using the subroutines

The subroutines described above are part of the MACRO program called "DTSUB.MAC". Once assembled, the object file for DTSUB (DTSUB.OBJ) may be linked with DIBOL programs that use the record locking or message facilities. An example of a LINK command is shown below.

```

.R LINK
*PROG=PROG,DTSUB,DIBOL

```

B.4 Miscellaneous functionsDetermining the TSX-Plus line number

The TSLIN subroutine can be called to determine the number of the TSX-Plus timesharing line from which the program is being run. Real lines are numbered consecutively starting at 1 in the same order they are specified when TSX-Plus is generated. Detached job lines occur next and virtual lines are numbered last.

The form of the call of TSLIN is:

```
XCALL TSLIN(lnum)
```

where "lnum" is a numeric variable capable of holding at least two digits into which is stored the TSX-Plus line number value.

Appendix C -- FILTIM PROGRAM

In addition to the date of creation of a file, TSX-Plus also stores file creation times in device directories. At the time a file is closed, the current time of day is automatically stored in the sixth word of the directory entry for that file. Under RT-11, this word is unused for permanent files and contains the job and channel number for tentative file entries. In order to represent the time as a positive 16-bit value, the time is converted to an integer representing the number of 3 second intervals since midnight. For example, if a file were closed at 11:13:22, then the sixth word of the permanent directory entry for that file would contain 13467 (32233 octal).

11 hr	*	60 min/hr	*	60 sec/min	/	3 sec/interval	=	13200
13 min			*	60 sec/min	/	3 sec/interval	=	260
22 sec					/	3 sec/interval	=	7

								13467

An EMT to obtain file directory information, including file creation times is provided by TSX-Plus. An EMT is also provided to set the file creation time value into file directory entries. See Chapter 7 for information on use of these EMTs.

The DIR utility provided with RT-11 does not interpret file creation time information as set by TSX-Plus, so a utility program (FILTIM) is provided with TSX-Plus to obtain this information. The FILTIM program should be copied from the distribution medium to the system device (SY:). It may then be run either explicitly (R FILTIM) or implicitly as a system program (FILTIM). Although FILTIM does not accept wildcard file specifications, it will accept up to 6 file specifications. The default device is "DK" and the default extension is "MAC". A device specification also applies to subsequent file specifications which do not explicitly include a device. Files created under RT-11 or before this feature was implemented in TSX-Plus will have a creation time of 00:00:00.

Example:

.FILTIM CKACT,TPRMAN.TXT,MAN:CH13.DPS,APPC.DPS,RK0:FILTIM				
DK:CKACT.MAC	3P	31-May-83	00:00:00	184
DK:TPRMAN.TXT	835	17-Jul-83	22:15:33	14226
MAN:CH13.DPS	35	18-Jul-83	08:31:42	3570
MAN:APPC.DPS	5	18-Jul-83	15:54:48	3605
RK0:FILTIM.MAC	23	18-Jul-83	11:24:57	4240

Note that the default device "DK" is used for CKACT and is carried over to the next file specification, and that the default file type is "MAC". The logical device "MAN" is used for the next two file specifications until the device specification, "RK0". And again, the default extension for RK0:FILTIM is "MAC".

Warning: Due to the method used by PIP for copy operations, file creation times are not preserved during copy operations, although the date is handled correctly. When copying a file with PIP, the destination file will acquire the time the copy is made as its creation time. The EMT to set file times, described in Chapter 7, may be used to set the correct time in the new file.

Appendix D -- RT-11 & TSX-Plus EMT CODES

D.1 TSX-Plus RT-11 Compatible EMTs

EMT	Code	Chan	Name	Description
340	-		.TTINR	Get character from terminal
341			.TTYOUT	Send character to terminal
342			.DSTATUS	Get device information
343	-		.FETCH/.RELEAS	Load/Unload device handlers
344			.CSIGEN	Call command string interpreter
345			.CSISPC/.GTLIN	Get command line
346	0		.LOCK	Lock USR in memory
347	0		.UNLOCK	Allow USR to swap
350			.EXIT	Return to monitor
351			.PRINT	Display string at terminal
352	-		.SRESET	Software reset
353	0		.QSET	Increase I/O queue size
354	-		.SETTOP	Set program upper limit
355			.RCTRLO	Reset CTRL-0
357	-		.HRESET	Stop I/O then .SRESET
374	0		.WAIT	Wait for I/O completion
374	1		.SPND	Suspend mainline program
374	2		.RSUM	Resume mainline program
374	3		.PURGE	Free a channel
374	4		.SERR	Inhibit abort on error
374	5		.HERR	Enable abort on error
374	6		.CLOSE	Close channel
374	0 7		.TLOCK	Try to lock USR
374	10		.CHAIN	Pass control to another program
374	* 11		.MWAIT	Wait for message
374	12		.DATE	Get current date
374	- 13		.ABTIO	Abort I/O in progress
375	0		.DELETE	Delete a file
375	1		.LOOKUP	Open existing file
375	2		.ENTER	Create file
375	3		.TRPSET	Intercept traps to 4 and 10
375	4		.RENAME	Rename a file
375	5		.SAVESTATUS	Save channel information
375	6		.REOPEN	Restore channel information
375	7		.CLOSE	Close channel
375	10		.READ[C][W]	Read from channel to memory
375	11		.WRIT[C][E][W]	Write from memory to channel
375	12		.WAIT	Wait for I/O completion
375	* 13		.CHCOPY	Open channel to file in use
375	14		.DEVICE	Load device registers on exit
375	- 15		.CDFN	Define extra I/O channels
375	- 20		.GTJB	Get job information

EMT Codes

EMT Code	Chan	Name	Description
375	21	.GTIM	Get time of day
375	22	.MRKT	Schedule completion routine
375	23	.CMKT	Cancel mark time
375	24	.TWAIT	Timed wait
375 *	25	.SDAT[C][W]	Send data to another job
375 *	26	.RCVD[C][W]	Receive data from another job
375	27	.CSTAT	Return channel information
375	30	.SFPA	Trap floating point errors
375 0	31	0 .PROTECT	Control interrupt vector
375 0	31	1 .UNPROTECT	Release interrupt vector
375	32	.SPFUN	Special device functions
375 *	33	.CNTXSW	Context switch
375	34	0 .GVAL	Get monitor offset value
375 -	34	1 .PEEK	Get low memory value
375	34	2 .PVAL	Change monitor offset value
375 -	34	3 .POKE	Change low memory value
375	35	.SCCA	Inhibit CTRL-C abort
375	36	0 .CRRG	Create an extended memory region
375	36	1 .ELRG	Eliminate an extended memory region
375	36	2 .CRAW	Create a virtual address window
375	36	3 .ELAW	Eliminate a virtual address window
375	36	4 .MAP	Map virtual window to XM region
375	36	5 .UNMAP	Get window mapping status
375	36	6 .GMCX	Obtain window status
375 *	37	0 .MTSET	Set terminal status
375 *	37	1 .MTGET	Get terminal status
375 *	37	2 .MTIN	Get character from terminal
375 *	37	3 .MTOUT	Send character to terminal
375 *	37	4 .MTRCTO	Reset CTRL-O
375 *	37	5 .MTATCH	Lock terminal to job
375 *	37	6 .MTDTCH	Release terminal from job
375 *	37	7 .MTPRNT	Display string at terminal
375 *	37	10 .MTSTAT	Get system status
375	40	.SDTIM	Set date and time
375	41	.SPCPS	Change mainline control flow
375	42	.SFDAT	Change file date
375	43	.FPROT	Change file protection

Notes:

- * Not supported, will cause error.
- 0 Treated as NOP.
- Minor differences, see Chapter 14.

D.2 TSX-Plus Specific EMTs

EMT	Code	Chan	Name	Description
375	101		UNLALL	Unlock all blocks
375	102		LOCKW	Wait for locked block
375	103		TLOCK	Try to lock a block
375	104		SNMSG	Send message on named channel
375	105		GETMSG	Get message from named channel
375	106		WATMSG	Wait for message on named channel
375	107	0	SPLFRE	Get number of free spool blocks
375	110	0	TSXLN	Get line number
375	111	0/1	ACTODT	Reset/set ODT activation mode
375	113		UNLOCK	Unlock a block
375	114	0	TTOBLK	Send block of text to terminal
375	115	0	TTIBLK	Get block of text from terminal
375	116	0	CKTTIE	Check for terminal input errors
375	117	0	SETTTO	Set terminal read time-out
375	120	0/1	HIEFF	Reset/set high-efficiency mode
375	121		CKWSHR	Check for writes to shared file
375	122		SAVSHR	Save shared file information
375	123	0	CKACT	Check for activation characters
375	125		SHRFIL	Declare file for shared access
375	127	0	SEND	Send message to another line
375	132	0	STRTDJ	Start a detached job
375	132	1	STATDJ	Check detached job status
375	132	2	ABRTDJ	Abort a detached job
375	133	0	DCLBRK	Establish break sentinel control
375	134	0	MOUNT	Mount a directory structure
375	135	0	DISMNT	Dismount a directory structure
375	136	0	INITPA	Initiate performance analysis
375	136	1	STRTPA	Start monitoring performance
375	136	2	STOPPA	Stop monitoring performance
375	136	3	HALTPA	Terminate performance analysis
375	137	0	TTYE	Get terminal type
375	140	0	PHYADD	Convert virtual to physical address
375	140	1	PEEKIO	Peek into the I/O page
375	140	2	POKEIO	Poke into the I/O page
375	140	3	BISIO	Bit-set into the I/O page
375	140	4	BICIO	Bit-clear into the I/O page
375	140	5	MAPIOP	Map PAR7 to the I/O page
375	140	6	MAPMON	Map PAR7 to simulated RMON
375	140	7	LOKLOW	Lock job into lowest memory
375	140	10	UNLOKJ	Unlock job from memory
375	140	11	ATTVEC	Attach to interrupt vector
375	140	12	RELVEC	Release interrupt vector
375	140	13	LOKJOB	Lock job without re-positioning
375	140	14	STEALS	Get exclusive system access

EMT Codes

EMT Code	Chan	Name	Description
375	140	15 RTURNS	Relinquish exclusive system access
375	140	16 SETPRI	Set user mode priority level
375	140	17 MAPPHY	Map to physical memory
375	140	20 ATTSVC	Attach interrupt service routine
375	140	21 SCHCMP	Schedule completion routine
375	141	0 MEMTOP	Control size of job
375	143	0 USERTS	Associate with run-time system
375	143	1 MAPRTS	Map run-time system into job
375	144	0 JSTAT	Get job status information
375	145	FILINF	Get file directory information
375	146	SFTIM	Set file creation time
375	147	0/1 GSUNAM	Get/set user name
375	150	0 SJBPRI	Set job execution priority
375	151	SPHOLD	Set spooler HOLD/NOHOLD
375	152	0 SELOPT	Select terminal option
375	153	0 NONINT	Set [non]interactive job status
375	154	0 STTSPD	Set line baud rate
375	155	0 GETCL	Assign CL unit to a line
375	156	0 ALCDEV	Allocate a device for exclusive use
375	156	1 DEALOC	Deallocate a device from exclusive use
375	156	2 TSTALC	Test device exclusive use allocation
375	157	0 MONJOB	Start monitoring job status
375	157	1 NOMONJ	Stop monitoring job status
375	157	2 STAMON	Broadcast status to monitoring jobs

Appendix E -- SUBROUTINES USED IN EXAMPLE PROGRAMS

E.1 PRTOCT - Print an octal value

The following subroutine accepts a value in R0 and prints the 6 digit octal representation of that value at the terminal.

```
.TITLE PRTOCT
.ENABLE LC

; Print octal value of the word in R0

.MCALL .PRINT
.GLOBL PRTOCT

PRTOCT: MOV     R1,-(SP)      ;Save R1-R3 on the stack
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     #EOW,R2     ;Point to end of 6 char output buffer
        MOV     #6,R3       ;Set up counter for 6 chars
1$:      MOV     R0,R1       ;Set up mask for low 3 bits (1s digit)
        BIC     #177770,R1  ;Get low 3 bits
        ADD     #'0,R1      ;Convert to ASCII
        MOV     R1,-(R2)    ;Fill octal digits in from end
        CLC
        ROR     R0
        ASH     #-2,R0      ;Shift out bits just converted
        SOB     R3,1$       ;Repeat for 6 chars
        .PRINT   #CHARS     ;Display result at console
        MOV     (SP)+,R3    ;Restore registers R1-R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        RETURN

CHARS:   .BLKB    6          ;6 char output buffer
EOW:     .ASCII   <200>     ;No CR terminator for .PRINT
        .EVEN
        .END
```

E.2 PRTDEC - Print a decimal value

The following subroutine accepts a value in R0 and prints the decimal representation of that value at the terminal with no leading zeroes.

```
.TITLE PRTDEC
.ENABLE LC

;
; Print the decimal value of the word in R0
;

.MCALL .PRINT
.GLOBL PRTDEC

;
PRTDEC: MOV     R1,-(SP)      ;Save R1 and R2
```


Subroutines for Examples

```

MOV      R2,-(SP)
MOV      #BUFEND,R2      ;Point to end of output buffer
MOV      R0,R1           ;Set up for DIV
1$:      CLR      R0      ;Clear high word for DIV
DIV      #10.,R0         ;Get least significant digit
ADD      #0,R1           ;Make remainder into ASCII
MOVB     R1,-(R2)        ;Save digit in output buffer
MOV      R0,R1           ;Set up for next DIV
BNE      1$             ;Until nothing left
.PRINT   R2              ;Display number at the terminal
MOV      (SP)+,R2        ;Restore R1 and R2
MOV      (SP)+,R1
RETURN
.BLKB    5               ;5 char output buffer
BUFEND:  .BYTE  200      ;No CR terminator for .PRINT
.EVEN
.END

```

E.3 PRTDE2 - Print a 2 digit decimal value

The following subroutine accepts a value in R0 and prints the decimal representation of it at the terminal. The value must be in the range of 0 to 99.

```

.TITLE   PRTDE2
.ENABLE  LC

; Print a 2-digit decimal value from R0

.MCALL   .PRINT
.GLOBL   PRTDE2

PRTDE2:  MOV      R1,-(SP)
MOV      R2,-(SP)
MOV      R3,-(SP)
MOV      R0,R1          ;Get copy of char in R1
MOV      #<BUFFER+2>,R2 ;Point to buffer
MOVB     #200,(R2)      ;Set end for .PRINT
MOV      #2,R3

2$:      CLR      R0      ;Clear high word for DIV
DIV      #10.,R0         ;Get low digit
ADD      #0,R1           ;Convert low digit to ASCII
MOVB     R1,-(R2)        ;Put digit in char buffer
MOV      R0,R1           ;Roll quotient for next DIV
SOB      R3,2$          ;Two digits only
.PRINT   #BUFFER        ;Display the result
MOV      (SP)+,R3
MOV      (SP)+,R2
MOV      (SP)+,R1
RETURN

```


Appendix E -- SUBROUTINES USED IN EXAMPLE PROGRAMS

E.1 PRTOCT - Print an octal value

The following subroutine accepts a value in R0 and prints the 6 digit octal representation of that value at the terminal.

```
.TITLE PRTOCT
.ENABLE LC

; Print octal value of the word in R0

.MCALL .PRINT
.GLOBL PRTOCT

PRTOCT: MOV     R1,-(SP)          ;Save R1-R3 on the stack
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     #EOW,R2         ;Point to end of 6 char output buffer
        MOV     #6,R3           ;Set up counter for 6 chars
1$:     MOV     R0,R1            ;Set up mask for low 3 bits (1s digit)
        BIC     #177770,R1      ;Get low 3 bits
        ADD     #0,R1           ;Convert to ASCII
        MOVB    R1,-(R2)        ;Fill octal digits in from end
        CLC
        ROR     R0
        ASH     #-2,R0          ;Shift out bits just converted
        SOB     R3,1$          ;Repeat for 6 chars
        .PRINT  #CHARS         ;Display result at console
        MOV     (SP)+,R3        ;Restore registers R1-R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        RETURN

CHARS:  .BLKB   6               ;6 char output buffer
EOW:    .ASCII  <200>          ;No CR terminator for .PRINT
        .EVEN
        .END
```

E.2 PRTDEC - Print a decimal value

The following subroutine accepts a value in R0 and prints the decimal representation of that value at the terminal with no leading zeroes.

```
.TITLE PRTDEC
.ENABLE LC

;
; Print the decimal value of the word in R0
;

.MCALL .PRINT
.GLOBL PRTDEC

;
PRTDEC: MOV     R1,-(SP)        ;Save R1 and R2
```


Subroutines for Examples

```

MOV      R2,-(SP)
MOV      #BUFEND,R2      ;Point to end of output buffer
MOV      R0,R1           ;Set up for DIV
1$:      CLR      R0      ;Clear high word for DIV
DIV      #10.,R0         ;Get least significant digit
ADD      #'0,R1          ;Make remainder into ASCII
MOVB     R1,-(R2)        ;Save digit in output buffer
MOV      R0,R1           ;Set up for next DIV
BNE      1$              ;Until nothing left
.PRINT   R2              ;Display number at the terminal
MOV      (SP)+,R2        ;Restore R1 and R2
MOV      (SP)+,R1
RETURN
.BLKB    5               ;5 char output buffer
BUFEND:  .BYTE  200      ;No CR terminator for .PRINT
.EVEN
.END

```

E.3 PRTDE2 - Print a 2 digit decimal value

The following subroutine accepts a value in R0 and prints the decimal representation of it at the terminal. The value must be in the range of 0 to 99.

```

.TITLE   PRTDE2
.ENABLE  LC

; Print a 2-digit decimal value from R0

.MCALL   .PRINT
.GLOBL   PRTDE2

PRTDE2:  MOV      R1,-(SP)
MOV      R2,-(SP)
MOV      R3,-(SP)
MOV      R0,R1           ;Get copy of char in R1
MOV      #<BUFFER+2>,R2  ;Point to buffer
MOVB     #200,(R2)       ;Set end for .PRINT
MOV      #2,R3

2$:      CLR      R0      ;Clear high word for DIV
DIV      #10.,R0         ;Get low digit
ADD      #'0,R1          ;Convert low digit to ASCII
MOVB     R1,-(R2)        ;Put digit in char buffer
MOV      R0,R1           ;Roll quotient for next DIV
SOB      R3,2$           ;Two digits only
.PRINT   #BUFFER         ;Display the result
MOV      (SP)+,R3
MOV      (SP)+,R2
MOV      (SP)+,R1
RETURN

```



```

BUFFER: .BLKB    6
        .EVEN
        .END

```

E.4 PRTR50 - Print a RAD50 word at the terminal

The following subroutine accepts a one word RAD50 value in R0 and prints its ASCII representation at the terminal.

```

        .TITLE  PRTR50
        .ENABL  LC
;
;  Display the RAD50 value of a word passed in R0
;
        .MCALL  .TTYOUT
        .GLOBL  PRTR50
PRTR50: MOV     R1,-(SP)          ;Need place to store last char
        MOV     R4,-(SP)          ;Need R4 and R5 for DIV
        MOV     R5,-(SP)
        MOV     R0,R5            ;Get copy of r0 into dividend
        CLR     R4                ;Zero high word for DIV
        DIV     #50,R4           ;Extract last char
        MOV     R5,R1            ;Save last char (remainder)
        MOV     R4,R5            ;Move quotient for next DIV
        CLR     R4                ;Zero high word for second DIV
        DIV     #50,R4           ;Get first and second chars
        .TTYOUT R5OTBL(R4)       ;Display first char (quotient)
        .TTYOUT R5OTBL(R5)       ; and second char (remainder)
        .TTYOUT R5OTBL(R1)       ; and last char
        MOV     (SP)+,R5          ;Restore registers
        MOV     (SP)+,R4
        MOV     (SP)+,R1
        RETURN
        .NLIST  BEX
R5OTBL: .ASCII  / ABCDEFGHIJKLMNOPQRSTUVWXYZ$. 0123456789/
        .EVEN
        .END

```

E.5 R5OASC - Convert a RAD50 string into an ASCIZ string

The following subroutine accepts a pointer in R0 to an argument block containing pointers to a RAD50 input string and an output buffer to hold the ASCII equivalent of the RAD50 string, and a word containing the number of characters to be converted.

Subroutines for Examples

```

        .TITLE   R50ASC
;
;   Convert RAD50 value into ASCII
;   Output buffer must be at least 3 characters extra long
;
        .GLOBL   R50ASC
        .DSABL   GBL
R50ASC: MOV      R1,-(SP)      ;Pointer to input buffer
        MOV      R2,-(SP)      ;Pointer to output buffer
        MOV      R3,-(SP)      ;Number of chars to convert
        MOV      R4,-(SP)      ;Dividend low word and quotient
        MOV      R5,-(SP)      ;Dividend hi word and remainder
        MOV      (R0)+,R1      ;Fetch input buffer address
        MOV      (R0)+,R2      ;Output buffer address
        MOV      (R0),R3       ;Number of chars to convert
        TST      R3           ;Any chars to convert?
        BLE      2$           ;Branch if not
1$:     ADD      #3,R2         ;Point to end of 3 char set
        MOV      (R1)+,R5      ;Fetch RAD50 value
        CLR      R4           ;Set up for divide
        DIV      #50,R4       ;Get first char
        MOVB     R50TBL(R5),-1(R2) ;Put 3/3 into output buffer
        MOV      R4,R5       ;Set up for divide
        CLR      R4
        DIV      #50,R4       ;Convert next char
        MOVB     R50TBL(R5),-2(R2) ;Put 2/3 into output buffer
        MOVB     R50TBL(R4),-3(R2) ;Put 1/3 into output buffer
        SUB      #3,R3       ;Converted 3 chars this round
        BGT      1$          ;Branch if more chars left to convert
2$:     ADD      R3,R2         ;Locate last real char
        CLRB     (R2)         ;Make output string ASCIIZ
        MOV      (SP)+,R5
        MOV      (SP)+,R4
        MOV      (SP)+,R3
        MOV      (SP)+,R2
        MOV      (SP)+,R1
        RETURN
        .NLIST   BEX
R50TBL: .ASCII   / ABCDEFGHIJKLMNOPQRSTUVWXYZ$. 0123456789/
        .EVEN
        .END

```

E.6 DSPDAT - Print a date value at the terminal

The following subroutine accepts a date value in R0 and prints the date representation at the terminal.

Subroutines for Examples

```
.TITLE DSPDAT
.ENABL LC
```

```
; Print a date value from R0
```

```
.MCALL .PRINT
.GLOBL DSPDAT,PRTDEC
```

```
MONMSK = 036000
DAYMSK = 001740
YRMSK = 000037
```

```
DSPDAT: MOV     R1, -(SP)
        MOV     R0, R1           ;Get copy of date in R1
        BIC     #^CDAYMSK, R0   ;Mask in only day bits (5-9)
        ASH     #-5, R0         ;Shift down
        CALL    PRTDEC          ;Print it out
        MOV     R1, R0         ;Get fresh copy of date
        BIC     #^CMONMSK, R0   ;Use only month bits (10-13)
        ASH     #-7, R0         ;Shift down to index into month table
        ADD     #MONTBL, R0     ;Point into table
        .PRINT  R0              ;Display the month
        MOV     R1, R0         ;Fresh copy again
        BIC     #^CYRMSK, R0   ;Use only year bits
        ADD     #72., R0        ;Year since 1972
        CALL    PRTDEC          ;Display the year
        MOV     (SP)+, R1
        RETURN
```

```
.NLIST BEX
MONTBL: .ASCII  /-NON-/<200><0><0>
        .ASCII  /-Jan-/<200><0><0>
        .ASCII  /-Feb-/<200><0><0>
        .ASCII  /-Mar-/<200><0><0>
        .ASCII  /-Apr-/<200><0><0>
        .ASCII  /-May-/<200><0><0>
        .ASCII  /-Jun-/<200><0><0>
        .ASCII  /-Jul-/<200><0><0>
        .ASCII  /-Aug-/<200><0><0>
        .ASCII  /-Sep-/<200><0><0>
        .ASCII  /-Oct-/<200><0><0>
        .ASCII  /-Nov-/<200><0><0>
        .ASCII  /-Dec-/<200><0><0>
        .EVEN
        .END
```


Subroutines for Examples

E.7 DSPTI3 - Display a 3-second format time value

The following subroutine accepts a special 3-second time value in R0 and prints the time value at the terminal.

```
.TITLE DSPTI3
.ENABL LC

; Display special 3-sec format time value from R0

.MCALL .TTYOUT
.GLOBL DSPTI3,PRTDE2

DSPTI3: MOV     R1,-(SP)
        MOV     R0,R1          ;Set up for divide
        CLR     R0
        DIV     #20.,R0        ;Get # of 3-SEC'S since midnight
        MOV     R1,-(SP)      ;Put on stack
        ASL     R1             ;2X 3-SEC'S
        ADD     R1,(SP)        ;Plus 1X gives 3X = seconds
        MOV     R0,R1          ;Get rest of time
        CLR     R0             ;Set up for next divide
        DIV     #60.,R0        ;Get number of minutes
        MOV     R1,-(SP)      ;And save on stack
        CALL    PRTDE2         ;What's left is hours, display
        .TTYOUT #' :
        MOV     (SP)+,R0       ;Recover minutes
        CALL    PRTDE2         ;Display minutes
        .TTYOUT #' :
        MOV     (SP)+,R0       ;Recover seconds
        CALL    PRTDE2
        MOV     (SP)+,R1
        RETURN
        .END
```

E.8 ACRTI3 - Convert a time value to special 3-second format

The following subroutine accepts a time value from the terminal and converts it to a special 3-second internal format. The value is returned in R0.

```
.TITLE ACRTI3
.ENABL LC

; Accept a time from the keyboard and return it in
; a special 3-second format in R0

.GLOBL ACRTI3

ACRTI3: MOV     R1,-(SP)
```


Subroutines for Examples

```

MOV     R2,-(SP)
MOV     R3,-(SP)
CLR     HOURS           ;Make sure it's reentrant
CLR     MINITS
CALL    GETNUM          ;Accrue decimal hours
BCS     2$              ;Return with error
MUL     #<60.*20.>,R3    ;Convert hours to 3-sec periods
MOV     R3,HOURS        ;Save hours in 3-sec units
TST     NUMERR          ;Did we hit end of input?
BNE     1$              ;Yes, return value
CALL    GETNUM          ;Accrue decimal minutes
BCS     2$              ;Return with error
MUL     #20.,R3          ;Convert minutes to 3-sec periods
MOV     R3,MINITS       ;Save minutes in 3-sec units
TST     NUMERR          ;End of input?
BNE     1$              ;Yes, return value
CALL    GETNUM          ;Accrue decimal seconds
BCS     2$              ;Return with error
CLR     R2              ;Convert seconds into 3-sec periods
DIV     #3,R2           ;Quotient stays in R2
1$:     ADD     MINITS,R2 ;Add in 3-secs from minutes
        ADD     HOURS,R2 ;Add in 3-secs from hours
        MOV     R2,R0    ;Return it in R0
        CLC         ;Say no error
        BR       3$      ;Return
2$:     SEC         ;Say there was an error
3$:     MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        RETURN

GETNUM: CLR     NUMERR   ;Say no error yet
        CLR     R3      ;Initialize number
1$:     MOVB    (R0)+,R1 ;Get next digit into R1
        CMPB    R1,#^0   ;Less than ^0?
        BLT     2$      ;Not a digit
        CMPB    R1,#^9   ;Greater than ^9?
        BGT     2$      ;Not a digit
        MUL     #10.,R3  ;Shift previous digits
        SUB     #^0,R1   ;Convert current digit to binary
        ADD     R1,R3    ;And include in number
        BR      1$      ;Get digits til next separator
2$:     CMPB    R1,#^:   ;Is it a legal separator?
        BEQ     3$      ;Yes, return
        INC     NUMERR   ;Say it may be end
        TSTB    R1      ;Was it a nul (end of input string)?
        BEQ     3$      ;Yes, return
        SEC         ;No, say invalid input
        RETURN        ;Error return
3$:     CLC         ;No error
        RETURN

```


Subroutines for Examples

HOURS: .WORD 0
MINITS: .WORD 0
NUMERR: .WORD 0
.END

Appendix F -- TSX-Plus USER ERROR MESSAGES

Several different categories of errors can generate messages, ranging from errors which are fatal to the operating system to something as simple as an incorrect file specification. Errors are identified by the name of the program which recognized the error and one or more lines of descriptive information. Errors identified by utility programs are identified by the name of the utility (e.g., ?PIP-F, ?DUP-F, ?KED-F) and may or may not abort the program. For more information about utility program error messages, consult the appropriate RT-11 manual (RT-11 System Message Manual, RT-11 System Utilities Manual, RT-11 Keypad Editor User's Guide, etc.). Errors which are fatal to the TSX-Plus operating system (identified as ?TSX-F) report error conditions for which time-sharing users are not usually responsible and usually cannot correct. TSX fatal errors should be reported to the system manager who should consult the TSX-Plus System Manager's Guide for more information. Errors which are not fatal to the operating system but which indicate a user mode error are reported as monitor errors. This Appendix describes TSX-Plus specific monitor error messages.

All user mode fatal monitor error messages have the syntax:

?KMON-F-" message "

or

?MON-F-" message "

where the "?KMON-F" form is used if the error occurs while using the keyboard monitor and the "?MON-F" form is used for errors which occur within a user program.

Warning messages have the form:

?KMON-W-" message "

In some cases, the message is only informational rather than indicating an error. In these cases, the message is of the form:

?KMON-I-" message "

Error messages are consistent with RT-11 usage insofar as possible. Monitor error messages unique to TSX-Plus are described below. Consult the RT-11 System Message Manual for monitor errors not described here.

Ambiguous command

Too few characters of a command keyword have been entered to allow the command to be uniquely identified. For example, this error would result if "CO" were entered as a command since CO could be an abbreviation for either COPY or COMPILE.

Ambiguous option

Not enough characters were specified to distinguish between options.

Error Messages

ASSIGN table full

Maximum number of logical assignments (15 per user) exceeded. SHOW ASSIGNS and DEASSIGN unnecessary logical assignments.

Attempt to increase MAXPRIORITY above current value

Maximum job priority may be restricted by the system manager. It may be reduced, but it may not be increased during a time-sharing session.

Cannot find spool file with specified ID/

There is no spool file in the queue with an ID number that matches the ID specified with the SPOOL command.

Cannot find SY:TSODT.REL file

The relocatable copy of the TSX-Plus ODT debugging program was not found on the system device. Copy the file "TSODT.REL" from the distribution medium to SY.

Cannot find SY:TSXUCL.SAV

A command was attempted to be interpreted as a user-defined command, but the TSXUCL program could not be found. Copy TSXUCL.SAV from the distribution to SY.

Cannot open alignment file

The alignment file specified in a SPOOL align command cannot be opened.

Cannot open logoff command file

A logoff command file which was specified during job start-up could not be found while logging the job off.

Cannot open spool device

The spooler cannot access the spool device. Verify the device name used in the SPOOL macro during TSX-Plus generation and check for correct device assignments.

Cannot open system accounting file

The file SY:ACCESS.TSX cannot be found during logon or logoff. This file is created by the system manager.

Closing log file

If a device on which a terminal output log file is open is initialized or squeezed, the log file is first closed.

Command file nesting too deep

Maximum depth of command file nesting exceeded. The depth of command file nesting allowed depends on the number and length of parameter strings. Three levels are possible even with long strings and as many as seven levels are possible with no parameters. See Chapter 3 for more information on command files.

Command file not found

The specified command file was not found. See Chapter 2 for an explanation of command interpretation including default devices, and Chapter 3 for more information on command files.

Command file parameter string too long

Total number of characters in parameter string exceeds the maximum allowed length of 60 characters. See Chapter 3.

Command string too complicated

The command string expansion is too large for the internal CCL buffer. Reduce the complexity of the command string. The SET CCL TEST command may be used to examine the expanded command string generated by high level commands.

Date unknown

The current date is not known. The date should be set by the operator by use of the DATE keyboard command.

Device full

There is insufficient free space available to open a file.

Device is allocated to another user

An attempt has been made to access a device that is currently allocated to another user by use of the ALLOCATE command.

Device is allocated to job nn

The device you are trying to access is allocated to job nn. The device must be deallocated before you may access it.

Device is being used by job nn

Job nn has a channel open to a device that you are trying to ALLOCATE, INITIALIZE, SQUEEZE, or SET. These commands may only be used when no one else is accessing the device.

Device must be allocated before use

The system manager has specified that the device you are attempting to access must be allocated to a job by use of the ALLOCATE command before it can be accessed by the job. Use the ALLOCATE command to allocate the device to your job.

Device is mounted by another job

The INITIALIZE and SQUEEZE commands are invalid if the device is MOUNTed by any other user. This minimizes the opportunity for data corruption when other users may be writing to a device. See the INITIALIZE and SQUEEZE commands for motivation.

Error Messages

Device is still mounted by other users

This informational message is displayed when a DISMOUNT request is issued for a device which is also MOUNTed by other users. See the INITIALIZE and SQUEEZE commands for motivation.

Device or file is access restricted

User does not have access privilege to requested device or file. The system manager may restrict user access to individual devices and files.

Directory I/O error

A device directory failed internal consistency checks. Devices must have valid RT-11 format directories for use with TSX-Plus. This error may also occur when attempting to use 22-bit addressing with DMA devices when the handler or hardware does not support 22-bit addressing.

File not found

The specified file was not found on the specified device.

Floating point trap

A floating point operation trap occurred and job had not done a .SFPA operation to specify a floating point trap processing routine.

Handler active -- Can't update running copy

An attempt was made to perform a SET command on a device handler which was in use. If the handler is idle when the SET is issued, the change will be made, otherwise, the running copy of the handler is not altered. Reissue the command when the handler is not active.

Illegal use of wildcards

Wildcards may not be specified for (part of) this command.

IND already active

The IND program cannot be run from within a command file which is being executed under control of the IND program.

IND is not available

The IND.SAV program was not on the system disk during TSX-Plus start-up. IND is provided with RT-11 version 5 and later.

Invalid address as EMT argument

An address specified in a monitor call was odd or was not within the job's address space. The value returned as the abort location is the address of the instruction or the EMT that caused the error.

Invalid boot device

An invalid device was specified with the BOOT command.

Invalid channel

An invalid channel number was used with an EMT.

Invalid command. Expected numeric value missing

A numeric parameter value is missing from a command that requires it.

Invalid command. Expected octal value missing

A numeric parameter (expressed in octal) is missing from a command that requires it.

Invalid date

An invalid value was specified as a date.

Invalid device

An invalid or unrecognizable device name was specified.

Invalid EMT

An invalid or unsupported function code was specified in an EMT argument block.

Invalid EMT argument

An invalid value was specified in an EMT argument block. Frequently this is caused by specifying an odd address where an even address is required -- such as for the address of a completion routine.

Invalid file name

An invalid file name was specified. Check for typing errors, names too long, and correct file specification format.

Invalid line number

An invalid line number was specified for a KILL or DETACH command.

Invalid logical disk name

Logical subset disks must be mounted as LDO through LD7.

Invalid multiple value on option

More than one parameter was specified for an option which only accepts one.

Invalid option

An invalid or unrecognized option keyword has been specified with a command.

Invalid or uninitialized directory

A disk device must contain a valid RT-11 format directory to be accessed by any operation other than INITIALIZE. Logical subset disks must be initialized before first use. This error may also occur when attempting to use 22-bit addressing with a DMA device and either the device handler or hardware does not support 22-bit addressing.

Invalid or unrecognizable device

A device name was specified which is not recognized by the system. This could happen if a logical device name was used without assigning the logical name to a physical device.

Error Messages

Invalid SAV file

A file specified with an R or RUN command failed internal consistency checks. The file may be damaged. An incorrect file type may have been specified - the default type is "SAV".

Invalid speed value

An invalid speed has been specified with a command used to set a line's transmit/receive speed. The valid speed values are: 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, and 19200.

Invalid start address for program

The start address for the program was outside of the program's memory bounds or on an odd address. A MACRO program may have an incorrect starting address specified with the .END directive. The file may be damaged. An incorrect file type may have been specified - the default is "SAV".

Invalid time

An invalid time value was specified with the TIME command.

Invalid unit number

An out of range unit number was specified with a SET CLn command. The valid unit numbers range from 0 (zero) up to the highest CL unit specified when the system was generated.

Kernel mode trap within TSX-Plus

A kernel mode trap error usually indicates the operating system image has been corrupted due to hardware or software failure. Record the full error message and report the error to the system manager. This error can also be generated by attempting to access a non-existent memory address with the real-time EMT's to access the I/O page.

Line is already being used as a CL unit

An attempt has been made to assign a CL unit to a line that already has another CL unit assigned to it. You may disassociate a CL unit from a line by assigning the CL unit to another line or to line 0 (zero) which disassociates it from any line.

Line is gagged

Messages cannot be sent to lines which have SET TT GAG unless those lines are waiting in TSKMON for a command.

Line is in use by a time-sharing user

An attempt has been made to assign a CL unit to a line on which a time-sharing user is logged in. The time-sharing user must log off before you can assign the CL unit to the line.

Line too long

The expanded CCL command was too long. Commands are expanded into the chain data area, locations 500 - 777. See the SET CCL TEST command.

Log file overflow

An error was reported while writing to the terminal output log file. This is most commonly an attempt to write past end-of-file. Make more room on the output device, specify a file size with the SET LOG command, or use the SET LOG NOWRITE option to reduce the volume of log file output.

Logical Disk support was not generated into system

Attempts to MOUNT a logical subset disk are not valid unless support for logical subset disks was selected when generating TSX-Plus.

Logical disks must be nested in order of increasing unit #

When a file residing within a logical subset disk is to be mounted as another logical subset disk, it must be assigned a higher number than the outer level logical subset disk. For example, the following command is invalid: MOUNT LD1 LD3:MYDISK; whereas the following is acceptable: MOUNT LD3 LD1:MYDISK.

Max allowed number of devices already mounted

The number of device directories which may be cached is defined during system generation. DISMOUNT devices on which caching is no longer needed or consult the system manager.

Memory size change not allowed in non-swapping TSX system

Jobs may not change size dynamically unless job-swapping was enabled during system generation. Consult the system manager.

Missing equal sign

The equal sign was missing on a SET device command.

No defined operator's console

No operator console was defined during system generation. Form mount requests and messages using the OPERATOR command are sent to the operator console.

No free detached job lines

The maximum number of detached job lines is specified during system generation. Either wait for a job to terminate, stop one by use of the DETACH/KILL command, or ask the system manager to increase the number of detached lines.

Not enough memory to run program

A file was too large (or the size specified in the SAV image was too large) to fit in the space allocated to the job. See Appendix A for more information on program size specifications. Use overlay segments or chain requests to reduce the program size. Use the MEMORY command to request more memory for your job. Note that you cannot use the MEMORY command in a non-swapping environment; see the system manager.

Error Messages

Performance monitor is in use by user number nn

The performance monitor facility is already in use. Only one job may use it at a time.

Priority value must be in the range ...

Job priority may not be set less than 0, nor higher than the maximum allowed priority.

Program debugger was not included when system was generated

A request was made to run a program with the TSX-Plus program debugging facility (i.e., RUN/DEBUG program) but the debugging facility was not included when the system was generated.

Prompt string too long

Prompt strings may not be longer than 8 characters.

Save file I/O error

An I/O error occurred while reading the SAV file. Check that the disk is still on line. It is also possible that the SAV file or the directory is damaged.

SL was not included at system generation

The SET SL ON command may not be issued unless support for the single line editor was selected when generating TSX-Plus.

System was not generated to support performance monitoring

Performance monitoring is an optional feature that must be included during system generation. See the system manager.

SL width is fixed at 80

Under TSX-Plus, the single line editor command buffer length is fixed at 80 characters and may not be changed with the SET SL width=nn command.

Table overflow

Too many devices/files have been specified in the ACCESS command. See the system manager.

Terminal type must be set to VT100, VT200 or VT52

The SET SL ON command may only be issued if the current terminal type is VT100, VT200 or VT52. Other terminal types are not supported by the single line editor.

This CL unit is currently busy

An attempt has been made to assign a CL unit to a line and the CL unit is currently busy processing an I/O operation.

This command only legal in startup command file

The ACCESS and SET LOGOFF commands are only valid in startup command files.

This operation not legal with SY (system) device

The device which was booted when starting TSX-Plus may not be initialized or squeezed while running TSX-Plus. If it is necessary to initialize or squeeze this device, do so while running RT-11.

This unit is not currently assigned to a line

A SET CL command has been issued to set some option for a CL unit and the CL unit is not assigned to a communications line.

Too many completion routines

You have attempted to connect too many completion routines to interrupt vectors. The number of interrupt vectors that can be connected to completion routines is determined during system generation. See the system manager.

Too many files

The command interpreter uses standard command string format: up to 6 input files and 3 output files. Reduce the number of file specifications accordingly.

Too many parameters to command file

TSX-Plus command files accept a maximum of six parameters. See Chapter 3.

Trap to 4

Abort location = nnnnnn

An execution trap has occurred to location 4. This trap is caused by (1) attempting to address an odd location by an instruction that requires an even address; or (2) attempting to access an invalid memory location (i.e., outside the valid range of addresses for your job).

Trap to 10

Abort location = nnnnnn

A reserved or unimplemented instruction has been executed. This can be caused by executing a floating point instruction on a machine that does not have a floating point unit.

Trap to 14

Abort location = nnnnnn

Invalid breakpoint trap executed. The abort location provided is the address of the instruction following the trap. In order to use breakpoint, the program must store the PC and PS for the breakpoint service routine in locations 14 and 16.

Trap to 20

Abort location = nnnnnn

Invalid IOT instruction executed. The abort location provided is the address of the instruction following the trap. In order to use an IOT instruction, a program must store the PC and PS of the IOT service routine in locations 20 and 22.

Error Messages

Trap to 34

Abort location = nnnnnn

Invalid TRAP instruction executed. The abort location provided is the address of the instruction following the trap. In order to use a TRAP instruction, a program must store the PC and PS of the TRAP service routine in locations 34 and 36.

TSGEN was modified without relinking TSKMON

Whenever a new TSX-Plus system generation is done, both TSX and TSKMON must be relinked. See the system manager.

Unable to allocate memory for virtual overlays

The system failed to successfully allocate extended memory regions when trying to run a program with virtual overlays. Unlock jobs from memory or increase the PLAS region swap file size.

Unable to open log file

The system reported an error when attempting to create a file for terminal output logging. Check the file specification, verify that there is room and that the output device is not write protected.

Unimplemented option

A command option has been specified which is not currently implemented by TSX-Plus.

Unrecognizable command

TSX-Plus could not find a system command, a user-defined command, or a command file with that name. See Chapter 2 for more information on command interpretation.

User command interface program not available

The system could not locate the file specified with the command: SET KMON UCI[=filnam]. Check the file specification. The default file is SY:UKMON.SAV.

USR called from completion routine

You cannot call system service routines that use TSUSR from a completion routine. This includes: .LOOKUP, .ENTER, .RENAME, .DELETE, .CLOSE, .DSTATUS, .SFDAT, and .FPROT.

USR err #n

All of the following USR errors result from consistency checks performed in closing a tentative file (creating a permanent file). These are usually indicative of a strange hardware condition, such as exchanging or squeezing a disk while a file is open.

Error Messages

USR err # 1

TSUSR can't find tentative file entry for specified file on close.

USR err # 2

File length in channel block is not equal to length in file entry.

USR err # 3

Highest block number written is greater than file length.

USR err # 4

Empty file entry doesn't follow tentative file entry.

USR err # 5

Tentative file entry status was lost during close operation. This is usually an indication of hardware failure.

Value required for option

A command was issued that required a value for an option, but no value was specified. Reissue the command correctly.

You are not authorized to write to that device or file

The system manager may restrict access to individual devices or files. See the system manager.

You are not privileged for that command

The command is a privileged command. The system manager may restrict the use of privileged commands.

You are not privileged to run this program

Operator privilege is required to run the program (e.g., SETUP).

?CCL-W-This command may interfere with other users

This warning is issued by CCL on INITIALIZE and SQUEEZE commands to warn you that other users using that device might be dismayed by what you are about to do. See the warning with the SQUEEZE command.

Appendix G -- LOGICAL SUBSET DISKS

Logical subset disks provide a method of logically partitioning a large disk into smaller units which can themselves be treated as directory structured devices. This is done by creating a (relatively large) file on a physical device and then creating a device directory and files within that file. The resulting pseudo device which is created within the file on the mother device is called a logical subset disk. Logical subset disks may also be nested. That is, one logical subset disk may be contained within another logical subset disk. This nesting may continue up to seven levels of logical subset disks within other logical subset disks. Logical subset disks may be initialized, squeezed, and have files created, opened, closed and deleted. They may also be assigned logical device names (e.g. OUT:, DK:, ABC:). Several keyboard commands are used to manipulate logical subset disks: DISMOUNT, MOUNT, SET LD and SHOW SUBSETS.

Support for logical subset disks is built into the kernel of TSX-Plus. Therefore, it is not necessary to use the LD device handler and logical subset disks may be used under TSX-Plus with either version 4 or version 5 RT-11 utilities.

Logical subset disks are given the device names LDO through LD7. Each user may use all eight logical subset disks at any time. That is, if one user has mounted LDO then any other user may also use LDO simultaneously and they need not (and usually will not) refer to the same file containing the logical subset disk. If logical subset disks are nested, then the device numbers must increase with the level of nesting. For example, LDO may contain a file to be mounted as LD1. However, a file contained within LD1 may never be mounted as LDO.

The typical sequence of operations when using logical subset disks is to: 1) create a file on a physical disk device; 2) mount that file as a logical subset disk; 3) initialize the new logical subset disk; 4) and then proceed to use it as any other disk device (except that it cannot be physically handled independently of the surrounding real disk). Subsequent uses of the logical disk only require that the disk be re-mounted. The file need not be re-created, nor the logical subset disk re-initialized.

The following example shows a typical sequence of commands which might be used to initiate use of a new logical subset disk.

```
.CREATE DL1:MYDISK.DSK/ALLOCATE:500.  
.MOUNT LDO: DL1:MYDISK  
.INITIALIZE/NOQUERY LDO:  
.  
.  
.  
.DISMOUNT LDO:
```

In order to use this new logical disk at a later time, it would only be necessary to issue the MOUNT command:

```
.MOUNT LDO: DL1:MYDISK DK:
```


LOGICAL SUBSET DISKS

The default (and recommended) file type for files intended to contain logical subset disks is .DSK.

When logical subset disks are mounted, information about them can be obtained with the SHOW SUBSETS command. For example:

```
.SHOW SUBSETS  
LD0 --> DL1:MYDISK.DSK[500]
```

To remove a logical subset disk from use, use the dismount command. For example:

```
.ASSIGN DL1 DK  
.DISMOUNT LD0:
```

Note that the ~~disk~~ files containing logical subset disks are automatically marked as protected files. In order to delete them, they must first be unprotected.

It is possible with logical subset disks to create some unusual situations and conflicts. For example, it is possible to mount a logical subset disk and then unprotect and delete the file which contained it. This condition would be marked with an asterisk (*) with the SHOW SUBSETS command. In addition, the SET LD CLEAN command can be used to force the system to compare, verify and correct the information in its internal logical subset disk tables if possible. Obviously, the system will not go back and recreate deleted files. An implicit SET LD CLEAN is done each time the DUP utility is called (e.g. with the DELETE and SQUEEZE commands).

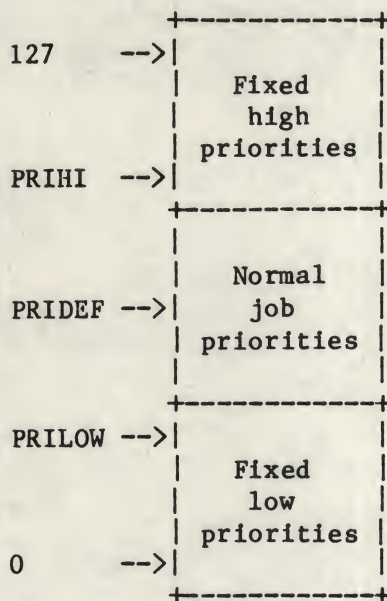
For more information on the details of the commands used with logical subset disks, see the descriptions in Chapter 2 of the following commands: DISMOUNT, MOUNT, SET LD CLEAN and SHOW SUBSETS.

Appendix H -- JOB EXECUTION PRIORITIES

TSX-Plus jobs may be assigned execution priorities to control their scheduling relative to other jobs. The priority values range from 0 to 127. The maximum execution priority that may be assigned to a job can be controlled by the system manager by use of the TSAUTH account authorization program or the SET MAXPRIORITY command (see the TSX-Plus System Manager's Guide).

The priority assigned to a job is set by use of the SET PRIORITY keyboard command or the TSX-Plus EMT described in Chapter 7. The current priority for a job and the maximum authorized priority can be displayed by use of the SHOW PRIORITY keyboard command, and may be obtained from within programs with the .GVAL request.

The priority values are arranged in three groups: the fixed-low-priority group consists of priority values from 0 up to the value specified by the PRILOW sysgen parameter; the fixed-high-priority group ranges from the value specified for the PRIHI sysgen parameter up to 127; the middle priority group ranges from (PRILOW+1) to (PRIHI-1). The following diagram illustrates the priority groups:



Job scheduling is performed differently for jobs in the fixed-high-priority and fixed-low-priority groups than for jobs with normal interactive priorities. Jobs with priorities in the fixed-low-priority group (0 to PRILOW) and the fixed-high-priority group (PRIHI to 127) execute at fixed priority values. That is, the priority absolutely controls the scheduling of the job for execution relative to other jobs. The job state does not influence the execution scheduling except as to whether the job is in a ready-to-run state or a wait state. A job with a fixed priority is allowed to execute as long as it wishes until a higher priority job becomes active.

The fixed-high-priority group is intended for use by real-time programs. The fixed-low-priority group is intended for use by very low priority background

Job Execution Priorities

tasks. Normal time-sharing jobs should not be assigned priorities in either of the fixed priority groups.

The middle group of priorities from (PRILOW+1) to (PRIHI-1) are intended to be used by normal, interactive, time-sharing jobs. Jobs with these assigned priorities are scheduled in a more sophisticated manner than the fixed-priority jobs. In addition to the assigned priority, external events such as terminal input completion, I/O completion, and timer quantum expiration play a role in determining the effective scheduling priority. For these jobs the job state is the primary factor in determining execution scheduling and the user-assigned job priority only influences the scheduling of jobs in the same state. See Chapter 5 of the TSX-Plus System Manager's Guide for further information about job scheduling.

When a job with a normal priority switches to a virtual line, the priority of the disconnected job is reduced by the amount specified by the PRIVIR sysgen parameter. This causes jobs that are not connected to terminals to execute at a lower priority than jobs that are. This priority reduction does not apply to jobs with priorities in the fixed-high-priority group or the fixed-low-priority group. The priority reduction is also constrained so that the priority of normal jobs will never be reduced below the value of (PRILOW+1).

Appendix I -- PROGRAM DEBUGGER

Programming errors can sometimes be very difficult to identify and correct without a detailed stepwise examination of a program's progress. Some higher level languages, like COBOL-Plus, include a debugging tool that allows such examination. For programs written in MACRO assembly language or when it is desirable to examine a program at the level of individual instructions, TSX-Plus provides a debugging facility with the following features:

1. The debugger is included as an optional system overlay and does not share memory space with the program being debugged. This allows programs up to the full 64Kb virtual address space to be run with the debugger. It can also be used with programs which are mapped to the I/O page or which use overlays, shared run-time systems, or extended memory regions (PLAS regions; virtual arrays or overlays).
2. The debugger does not have to be linked with the program being debugged. It can be invoked when the program is executed by the RUN/DEBUG command.
3. A keyboard command can be issued to permit debugging of any subsequent user program or utility even if the program is not started with the RUN/DEBUG command.
4. Execution of a program can be interrupted at any time by typing control-D. This causes entry to the debugger similar to hitting a breakpoint in the program.
5. The debugger can decode instructions into symbolic assembly language.
6. Program traps to 4 and 10 are caught by the debugger rather than causing a program abort with return to the keyboard monitor.
7. A data "watchpoint" facility is included which can cause an execution break to occur when the value of a monitored data item is changed.

I.1 Debugger requirements

In order to make effective use of the debugger facility, it is necessary to have a link map of the program being debugged. This minimizes the usefulness of the debugger for interpreted languages (e.g. BASIC) and languages with a run-time system (e.g. DIBOL) unless one is interested in debugging the run-time interpreter. For other high-level languages (e.g. FORTRAN and Pascal) the debugger is most useful for small program sections for which the intermediate assembly code is available.

The debugger facility is an optional feature of TSX-Plus, included during system generation. Debugger support must have been selected during system generation (DBGFLG = 1) in order to use this feature.

Program Debugger

I.2 Invoking the debugger

The debugger need not be linked with the program being debugged. Programs may either be started under debugger control with the RUN/DEBUG command, or can be interrupted while running by typing control-D if the SET CTRLD DEBUG command has been previously issued.

I.2.1 RUN/DEBUG switch:

A program may be started under control of the debugger with the /DEBUG switch to the RUN command. For example:

.R/DEBUG PRGNAM

When a program is started under debugger control, the program is loaded and the debugger prompt appears. For example:

.RUN/DEBUG PRGNAM

TSX-Plus debugger

DBG:

Whenever the debugger is in control, the "DBG:" prompt will appear and the debugger will be ready to accept a command.

On entry to the debugger, the program start address is loaded into R0 and set so that program execution may be initiated with the ";G" command without specifying a starting address. It is usually desirable to set initial values for relocation registers and breakpoints before starting program execution.

I.2.2 Control-D break

Control-D normally only interrupts program execution if the program has been started by the RUN/DEBUG command. However, a keyboard command is available to enable control-D to interrupt a program and pass control to the debugger even if the program was not started under debugger control. (See the description of the SET CTRLD DEBUG command in chapter 2.) If control-D interruption has been enabled, then any program may be interrupted and control transferred to the debugger regardless of whether the program was started under debugger control. The effect is similar to hitting a breakpoint at the current location. The same debugger commands and facilities are available when the debugger is entered this way as when the program is started under debugger control. Note however, that if the SET CTRLD DEBUG command has not been issued prior to starting a program's execution and that program is not started with the RUN/DEBUG command, then control-D will have no special effect on the program and debugger control cannot be obtained without restarting the program. The SET CTRLD DEBUG command is local to each time-sharing line and must be invoked from that line to enable control-D breakpoints in programs which are not started with the RUN/DEBUG command. If a program is started with the RUN/DEBUG command, then control-D may be used to interrupt its execution regardless of whether the SET CTRLD DEBUG command has been issued.

See the "Special notes" section at the end of this appendix for information about interactions of control-D with special terminal mode and special activation characters.

I.3 Commands

The commands and syntax of the debugging facility are similar to those of RT-11's ODT. However, not all ODT commands are implemented and some additional features are provided. In the following list of debugger commands, angle brackets are used to indicate special terminal keys such as <RETURN>, <TAB> and <LINE FEED>. The angle brackets are not part of debugger command syntax. Many commands accept a numerical value or address, indicated here as value, address, repeat or n. The underlined portion should be substituted with the appropriate number when issuing the command. The value is optional in some cases. All numerical values are assumed to be octal unless terminated by a decimal point.

Debugger Commands

<u>address</u> /	Display the contents of the addressed word.
<u>address</u> \	Display the contents of the addressed byte.
<u>value</u> <RETURN>	Store <u>value</u> into the currently open cell. If <u>value</u> is blank, close the current location without changing its contents.
<u>value</u> <LINE FEED>	Store <u>value</u> into the currently open cell and then advance to the next word or byte. If <u>value</u> is blank, advance to the next location without changing the value of the current location.
<u>value</u> ^	Store <u>value</u> into the currently open cell and then advance to the previous word or byte. If <u>value</u> is blank, back up to the previous location without changing the contents of the current location.
<BACKSPACE>	Equivalent to up-arrow ("^").
@	Open the cell whose address is specified by the contents of the currently open cell.
_ (underscore)	Open the cell whose address is specified by the contents of the currently open cell plus the current location plus 2.
<u>address</u> [Open the cell whose address is specified and display its contents as a symbolic instruction. If <u>address</u> is blank, decode the contents of the currently open cell into symbolic instruction form.

Program Debugger

address]

Open the cell whose address is specified and display its contents as a symbolic instruction. If address is blank and a cell is currently open as an instruction, close the current cell and open the next cell and display its contents in symbolic instruction format. The location of the next cell opened is determined by the number of words used by the instruction in the currently open cell.

value<TAB>

Execute the next instruction in single step mode. It is not necessary to issue the ";1S" command to single step using the TAB key. If value is non-blank, the value is used to calculate the address of the next instruction to be single stepped (similar to ;1S, then value;P). If 0 (zero) is substituted for value before <TAB> is typed and the next instruction is a JSR (CALL), then a temporary breakpoint is set beyond the JSR to catch the return from the called subroutine and the subroutine is executed without single stepping. This can be used to avoid single stepping through subroutines called from code that is being single stepped.

address;nR

Set relocation base register n to address. There are 8 relocation registers, numbered 0 through 7.

nR

Convert the contents of the currently open cell into an offset relative to relocation register n.

n!

Convert the current address to an offset relative to relocation register n.

address;nB

Set an instruction breakpoint at address. There are 8 instruction breakpoints, numbered 0 through 7. To clear a breakpoint, substitute 0 for address.

address;G

Go to location address and start program execution there.

repeat;P

Proceed from a breakpoint. If repeat is non-blank, do not break again until repeat breakpoints have been hit. If repeat is blank, proceed until the next breakpoint is hit.

;nS

Set or reset single stepping mode. If n is 1 (or any other non-zero value), begin single step mode. When single stepping, execution will proceed one instruction for each execution of the ";P" command. If n is 0 or is blank, single stepping mode is cancelled and the program will resume normal execution when the ";P" command is issued. (See also the <TAB> command.)

X

Interpret the contents of the currently open word as a RAD50 value and display it as a three character ASCII string. Up to three characters may be entered after the current value is displayed. The new characters will be converted to a RAD50 value and stored in the current location. Any characters beyond the first three are ignored.

address;valueM

Set a data watchpoint. The word specified by address is monitored. If value is specified, a data watchpoint occurs when the value of the word matches value. It is possible to "watch" only selected bits of the monitored address by setting the mask register (see \$M and data watchpoint description below). If value is blank, a data watchpoint occurs any time the value of the word changes.

I.4 Control-D breakpoints

When a program has been started by the RUN/DEBUG command, its execution can be interrupted at any time by typing control-D. (If the SET CTRLD DEBUG command has been issued, then the program need not have been started under debugger control.) Typing control-D causes the program to stop as though a breakpoint had been encountered at the current location and passes control to the debugger. Any valid debugger command may be issued at this point, and program execution may be resumed with the ";P" command. If a system service call (EMT) is being executed when control-D is typed, the service call will run to completion and control will be passed to the debugger at the point of return to the program from the EMT.

See the "Special notes" section at the end of this appendix for more information on interactions of control-D with special terminal mode and special activation characters.

I.5 Address relocation

Address values in commands may be specified absolutely or in the form "n,offset", where n is a relocation register number and offset is the offset relative to that relocation register.

When an instruction is decoded into symbolic form, relative addresses are displayed in the form "[n,offset]" as an offset from the nearest relocation register whose value is lower than the address. The format register (\$F) can be used to control the display format of addresses.

Program Debugger

Only two bits are significant in the \$F register, bits 0 and 1. Bit 0 controls the display of instruction address operands, and bit 1 controls the display of location values. Setting either bit causes the corresponding addresses to be displayed in absolute format. Clearing either bit causes the corresponding addresses to be displayed in relative format. Both default to relative format (\$F = 0).

Address Display Format

<u>\$F</u>	<u>Locations</u>	<u>Operands</u>
0	relative	relative
1	relative	absolute
2	absolute	relative
3	absolute	absolute

I.6 Internal registers

The following special address symbols are used to specify machine registers:

\$0 through \$5 -- Registers R0 through R5.

\$6 -- Register 6, the stack pointer (SP).

\$7 -- Register 7, the program counter (PC).

\$S -- Processor Status Word.

The following special address symbols are used to specify internal debugger registers:

\$M -- Mask register, used with data watchpoints (see below).

\$F -- Print format control register (see above).

\$B -- Start of instruction breakpoint address registers.

\$R -- Start of relocation registers.

I.7 Data watchpoints

The data watchpoint facility is a powerful tool for determining where a particular data cell is being modified. The form of the command used to initiate a watchpoint is:

address;valueM

where address is the address of the word that is to be monitored, and value is an optional parameter that specifies a value to be watched for. If value is blank, a data watchpoint occurs any time the value of the word changes. If value is specified, a watchpoint only occurs when the value of the word matches the specified value. When a watchpoint occurs, the program counter is pointing past the instruction which modified the word.

A data mask may be specified to select a portion of the word being monitored with the watchpoint. The mask value may be referenced using the register name "\$M". Initially the value of the mask is 177777 which causes the entire word contents to be monitored. If some other value is stored in \$M, only the bits selected by the mask are monitored.

Use of the data watchpoint facility causes the program to execute very slowly. When a watchpoint is in effect the debugger causes a break to occur on each instruction executed and then checks to see if the monitored data word has changed value. If a watchpoint is in effect but no instruction breakpoints are set, the execution of the program is slowed down by a factor of about 55 (i.e., the program takes 55 times as long to execute). If a watchpoint is in effect and instruction breakpoints are also set, the speed reduction factor is about 80. There is no significant speed reduction when instruction breakpoints are set but no watchpoint is in effect. The best strategy is to set an instruction breakpoint as close as possible to the code which is modifying the data word and then, when that breakpoint is reached, remove the instruction breakpoint, set the data watchpoint, and proceed with execution.

I.8 Symbolic instruction decoding

Symbolic instruction decoding is used to interpret values according to their numerical op codes and convert them into symbolic assembly language format. The display format of address operands is controlled by the \$F register. When a breakpoint is reached, the instruction on which the breakpoint was set is displayed in its symbolic form. Specific locations can also be symbolically decoded with the "[" and "]" operators.

The "[" command can be issued to symbolically decode a value which has just been displayed using the "/" command or to symbolically decode the contents of a specific location. The following examples illustrate how this might be done (the commands typed by the user are underlined):

```
DBG:2030/ 010246 [ MOV      R2,-(SP)
DBG:3000[    BIT      #2000,[3,100]
```

The "]" operator closes the currently open cell, opens the next cell and displays its contents in symbolic instruction form. The number of words to skip from the currently open cell to the next cell is determined by the number of words used by the instruction in the currently open cell. Thus, the "]" operator is convenient for examining a consecutive set of instructions.

Program Debugger

I.9 Special notes

Since control-D is used by the debugger to dynamically interrupt a program, control-D will not be passed through to a running program even if the program is in special terminal input mode (TTSPC\$ bit set in the JSW). If the program is waiting for terminal input when control-D is typed, then it is necessary to type an activation character (usually <RETURN>) before causing a break to occur and passing control to the debugger.

If a program declares control-D as a special activation character, then control-D is always passed to the program and cannot be used to cause a break in that program and pass control to the debugger.

When the debugger is operating in single step mode, normal breakpoints are temporarily removed and are restored when leaving single step mode. A side effect of this is that if the program executes an instruction (such as RTT) which clears the T (trace) bit in the processor status word, then single stepping is lost, breakpoints are not restored and the program continues execution.

Breakpoints set in overlay regions are only valid while that overlay is resident. If another overlay becomes resident, any breakpoints set in the previous overlay are lost and must be reset.

When examining the processor status word (\$S register), keep in mind that only the low 4 bits (condition codes) are significant to the program. The trace bit is used by the debugger. The priority bits are not significant in user mode under TSX-Plus. And, the high 4 bits will always be set, indicating user as both previous and current mode.

Index

- .ABTIO, 245
- .CDFN, 245
- .CHAIN, 245
- .CHCOPY, 245
- .CNTXSW, 245
- .CSIGEN EMT, 90
- .CSISPC EMT, 90
- .DEVICE, 245
- .DEVICE EMT, 214
- .FETCH, 245
- .FORK, 245
- .GTJB, 245
- .GTLIN EMT, 90
- .GVAL EMT
 - Checking I/O page mapping, 198
 - Special TSX-Plus use, 126
- .HRESET, 245
- .INTEN, 245
- .LOCK, 245
- .MTPS, 246
- .MWAIT, 246
- .PEEK, 201, 246
- .POKE, 201, 246
- .PROTECT, 230, 246
- .PURGE
 - Shared files and, 175
- .QSET, 246
- .RCVD, 246
- .RELEAS, 246
- .RSUM, 212
- .SAVESTATUS
 - Shared files and, 175
- .SDAT, 246
- .SDTTM, 246
- .SETTOP, 125, 246, 251
- .SFPA, 246
- .SPND, 212
- .SRESET, 246
- .SYNCH, 247
- .TLOCK, 247
- .TTINR, 247
- .TTYIN EMT
 - Command file input, 90
 - Non-wait input, 70, 119
 - Time-out value, 143
- .UNLOCK, 247
- .UNPROTECT, 247
- 8-bit terminal I/O, 67
- ?INI-E3, 248
- ABORT command, 86
- Aborting jobs
 - Detached jobs, 97
 - Normal jobs, 38
- ACCESS command, 28
- ACRTI3, 274
- Activation characters, 111
 - Checking for, 147
 - Control-D, 300
 - Defining, 117
 - Field width, 118
 - ODT activation mode, 159
 - Resetting, 118
 - Time-out activation, 143
- Adapting RT-11 Real-time programs, 230
- Administrative control, 3
- Alignment of forms, 109
- ALLOCATE command, 28
- Allocated devices
 - SHOW command, 73
- Allocating devices, 156
- ASSIGN command, 29
- Assigns
 - Displaying those in effect, 73
- B(ASE) command, 86
- Backing up in a spool file, 105
- BACKUP command, 30
- Basic operation, 9
- BATCH facility, 247
- Baud rate
 - Selection, 69
 - Setting, 151
- Block locking
 - See Shared files
- BOOT command, 31, 86
- Booting the system, 85
- Break sentinel control, 144
- BYE command, 31
- CACHE
 - SHOW command, 73
- CACHE parameter
 - Selecting appropriate size, 49
- Caching
 - Data, 184
 - Directories, 41
- Carriage-return
 - Automatic line-feed, 120
- Cataloged procedures
 - See Command files
- Channel numbers, 248, 249

Index

- Chapter summaries, 4
- Character echoing, 66, 67, 117
- CL device
 - VTCOM, 53
- CL units, 3
 - Assigning, 53, 152
 - SET options, 50
 - SHOW command, 73
 - Using with VTCOM, 51
- CLOSE command, 86
- COBOL command, 31
- Command file control, 87
- Command files, 87
 - Comments within, 89
 - Control characters within, 90
 - Controlling input, 90, 118
 - Controlling listing, 68, 89, 90
 - Displaying messages, 91
 - Invoking, 17, 18, 88
 - Nesting of, 89
 - Parameter strings, 88
 - PAUSE command, 44
 - Pausing execution, 91
 - Setting error abort level, 56
- Command interpreter, 17
- Commands
 - Defining, 20
 - Listing user-defined, 74
- Common data areas, 233
- Communication units
 - SET options, 50
- COMPILE command, 32
- Completion routine
 - Connecting to an interrupt, 224
 - Scheduling, 229
 - Scheduling message, 192
- Configuration
 - SHOW command, 74
- Configuration requirements, 1
- Control characters, 11
 - Ctrl-C, 11, 117
 - Ctrl-D, 11, 55, 293, 294, 300
 - Ctrl-O, 11
 - Ctrl-Q, 11, 69
 - Ctrl-R, 11
 - Ctrl-S, 11, 69
 - Ctrl-U, 11
 - Ctrl-W, 11, 93
 - Ctrl-Z, 11
 - Within command files, 90
- Control files
 - See Command files
- Control-D breakpoints, 297
- Cooperative file access
 - See Shared files
- COPY command, 32
- CORTIM parameter
 - Setting value, 54
 - SHOW command, 75
- CREATE command, 32
- CRT terminal support, 68
- D(EPOSIT) command, 86
- Data caching, 184
 - Enabling use of, 172
 - Setting number of cache buffers, 61
 - Suppression of, 184
- Data terminal ready
 - CL device, 52
- Data watchpoint
 - Debugger, 293
- DATE command, 32
- DBL default compiler, 59
- DEALLOCATE command, 32
- DEASSIGN command, 33
- Debugger, 293
 - Address relocation, 297
 - Breakpoints, 296
 - Control-D, 300
 - Control-D breakpoints, 55
 - Data watchpoints, 297, 298
 - Features, 293
 - Format register, 298
 - Internal registers, 298
 - Mask register, 299
 - Overlays, 300
 - RAD50 values, 297
 - Relocation base, 296
 - Single stepping, 296, 300
 - Special activation characters, 300
 - Special terminal mode, 300
 - Speed, 299
 - Symbolic decoding, 295, 299
 - System generation, 293
- Debugging programs, 45
- Deferred character echoing, 66
- DELETE command, 33
- DELETE key
 - Rubout filler character, 116

- DETACH command, 33, 95, 96, 97
- Detached jobs, 93, 94
 - Aborting, 34, 97, 99
 - Checking status, 34
 - Checking status of, 96, 100
 - Comparison with virtual lines, 95
 - Control EMTs for, 97
 - Keyboard control commands, 33
 - Starting, 33, 95, 97
- Device allocation, 28, 156
 - SHOW command, 73
- Device spooling
 - See Spooling
- DIBOL, 248
 - Record locking, 248
 - SEND statement, 248
- DIBOL command, 35, 86
- DIBOL default compiler, 59
- DIBOL record locking procedures, 171
- DIBOL support subroutines, 255
- DIFFERENCES command, 35
- Directory caching, 41
 - Dismounting a file structure, 142
 - Displaying mounted devices, 77
 - Mounting a file structure, 140
 - SQUEEZE command effect, 82
- DIRECTORY command, 35
- Directory information
 - EMT to obtain, 161
- DISMOUNT command, 35, 289
- Dismounting a file structure, 142
- DISPLAY command, 36, 91
- DL-11, 1
- DSPDAT, 272
- DSPTI3, 274
- DTR
 - CL device, 52
- DU
 - Partitioning, 248
- DUMP command, 36
- DZ-11, 1
- E(XAMINE) command, 86
- Echo control, 67, 117
- EDIT, 55
- EDIT command, 36
- Editor
 - Selecting default, 55
 - Single line, 12
- Eight-bit terminal I/O, 67, 70
- EMT codes
 - Table, 265
- EMT differences, 245
- EMT tracing, 55
- EMT's
 - TSX-Plus specific, 121
- Error messages, 277
- Escape character
 - Within command files, 90
- Escape sequence processing, 116
- Escape sequences
 - CL device, 52
- Exclusive access to a file, 171
- Exclusive system control, 209
 - Releasing, 209
- EXECUTE command, 37
- Execution priority, 291
 - Virtual lines, 93, 94
- Extended memory regions, 167
- Field width activation, 118
- Field width limit for TT input, 120
- File
 - Block locking, 171
 - Data caching, 184
 - Exclusive access, 171
 - Opening for shared access, 171
 - Protection modes, 171
 - Shared access, 171
- File creation time, 163
- File directory information, 161
- Fixed-high-priority
 - Determining, 126
- Fixed-low-priority
 - Determining, 126
- Floating point, 246
- Foreground programs
 - See Real-time support
- Form alignment procedure, 109
- FORM command, 37, 108
- Form feed control, 67
- Form names, 103, 108
- FORMAT program, 247
- FORTRAN
 - I/O page mapping, 249
 - Virtual arrays, 230, 248
- FORTRAN command, 37

Index

- FRUN command, 86
- Generalized data cache
 - Selecting appropriate size, 49
- GET command, 86
- Global data areas, 233
- GT ON/OFF command, 86
- Hardware requirements, 1
- Hazeltine terminal support, 67
- HELP command, 37
- High-efficiency terminal mode, 45, 114, 115, 118, 157
- HIPRCT parameter
 - Setting value, 56
- SHOW command, 76
- I/O channels, 248, 249
- I/O page
 - Accessing, 169, 198
 - FORTTRAN, 249
 - Mapping, 45
- IND
 - .ASK directive, 248
 - Command file aborting, 56, 57
 - Command file processing, 57
 - Control of command files, 87
- INDEX, 304
- INI-E3, 248
- INITIALIZE command, 37
- INSTALL command, 86
- Interactive jobs
 - Selecting dynamically, 138
- Interprogram communication, 187
 - Checking for messages, 189
 - Common memory areas, 233
 - Message channels, 187
 - Sending a message, 187
 - Waiting for a message, 190
- Interrupt completion routine, 224
- Interrupt processing
 - (Diagram), 218
- Interrupts
 - Connecting to real-time jobs, 216
- INTIOC parameter, 57
 - SHOW command, 76
- Introduction, 1
- IO abort handling, 57
- Job monitoring, 131
- Job number
 - Determining, 126
- Job priority
 - Determining, 126
 - Maximum, 60
 - Setting, 61, 136, 215
 - SHOW command, 78
- Job scheduling, 2
- Job status information, 128
- Job status word
 - Non-wait TT input, 70, 119
 - Setting virtual flag with SETSIZ, 253
 - Virtual-image flag, 167
- K52, 55
 - Virtual lines, 94
- KED, 55
 - Virtual lines, 94
- Keyboard commands, 17, 28
 - Abbreviation of, 17
- Keyboard monitor
 - Prompt character, 62
- KILL command, 38
- KJOB command, 39
- LA120 terminal support, 67
- LA36 terminal support, 67
- LD handler, 247, 289
- Lead-in character, 113, 115
 - Determining, 126
- LIBRARY command, 39
- License number
 - Determining TSX-Plus, 126
- Line number
 - Determining, 122
 - Determining primary, 126
- Line-feed
 - Echoing of, 120
 - Ignored with tape mode, 69, 120
- Lines
 - Assigning to CL, 152
- LINK command, 39
 - /XM switch, 167
- Link map, 293
- LOAD command, 86
- Locking a form on a spooled device, 103
- Locking a job in memory, 210
- Log off command files, 60
- Logging off, 10, 43
- Logging on, 9
 - Password, 9
 - Project programmer number, 9

- User name, 9
- Virtual lines, 93
- Logging terminal output, 59
- Logical device names, 29
 - Assigning, 73
- Logical subset disks
 - Dismounting, 35
 - Mounting, 42
 - Nesting, 289
 - Using, 289
 - Verifying, 59
- LOGOFF command, 39
- Lower-case character input, 68, 117
- MACRO command, 39
- MAKE command, 39
- Mapping virtual region, 206
- Maximum priority
 - Determining, 126
 - Setting, 60
- Memory
 - Using as pseudo-disk, 169
- Memory allocation
 - EMT to control, 125
 - MEMORY command, 40
 - Setting size in SAV file, 251
- MEMORY command, 40, 126, 251
- Memory mapping, 165
- Memory space
 - Displaying value, 76
- Message channels, 187
 - Completion routine, 192
- Message communication
 - See Interprogram communication
- Messages
 - Error, 277
 - Inhibiting, 67
 - Sending to another line, 140
- MicroPower/Pascal, 249
- Modem control, 68
- Modification of shared files
 - Checking for, 183
- MONITOR command, 41, 237
- Monitoring another job, 131
- MOUNT command, 41, 140, 289
- Mounting a file structure, 41, 140
- Multi-terminal EMTs, 245
- MUNG command, 43
- Non-interactive jobs
 - Selecting dynamically, 138
- Non-wait TT input, 70, 119
- Normal programs, 167
- NUMDC parameter
 - Setting value, 61
 - SHOW command, 77
- Obtaining TSX-Plus system values, 126
- ODT, 247
 - See Debugger
- ODT activation mode, 159
- ODT debugger, 247
- OFF command, 43, 94
- Opening shared files, 171
- OPERATOR command, 43
- Operator communication, 43
- Operator privilege, 68
 - Determining, 126
 - Real-time jobs, 197
- Optimization
 - SET SIGNAL, 63
- Overlaid programs, 249
- Page length control, 68
- Paint character
 - See Rubout filler character
- Paper tape mode
 - See Tape mode
- PAR 7
 - FORTTRAN, 249
- PAR 7 mapping, 45
 - Determining, 126
- Parameter strings for command files, 88
- Password
 - Changing, 9
 - Logging on, 9
- PAUSE command, 44, 91
- Performance monitor, 237
 - Control EMT's, 239
 - Displaying results, 238
 - MONITOR command, 41, 237
 - Starting, 237, 239
- Physical address calculation, 212
- Physical address space, 165
- Physical memory access, 206
- PLAS support, 167
- Poke EMT, 203
- Primary line
 - Determining, 126
- PRINT command, 44

Index

- Printer form names, 103
- Priorities, 291
- Priority
 - Maximum, 60
- Priority level
 - Setting, 61, 136, 214
- PRIORITY parameter
 - Setting value, 61
 - SHOW command, 78
- Privilege
 - Operator, 68
- PRIVIR parameter, 137, 216, 292
- Processor status word
 - Debugger, 300
- Program controlled terminal options
 - See Terminal control
- Program debugger, 293
- Programmed Logical Address Space
 - See PLAS support
- Programmer number
 - Determining, 126
- Project number
 - Determining, 126
- PROTECT command, 44
- Protected access to a file, 171
- PRTDE2, 270
- PRTDEC, 269
- PRTUCT, 269
- PRTR50, 271
- Pseudo-disk in memory, 169
- QUAN values
 - Selecting, 63
- QUANxx parameters
 - Setting value, 62
 - SHOW command, 78
- QUEMAN, 247
- QUEUE, 247
- R command, 44
 - /DEBUG switch, 45, 126
 - /HIGH switch, 45
 - /IOPAGE switch, 45
 - /LOCK switch, 46
 - /NONINTERACTIVE switch, 46
 - /SINGLECHAR switch, 46, 119
 - Implicit, 18
- R50ASC, 271
- Read time-out value for TT inputs, 143
- Real-time completion routine, 224
- Real-time jobs
 - Operator privilege, 197
- Real-time priority, 225
- Real-time support, 197
 - Accessing the I/O page, 198, 201, 206
 - Adapting RT-11 programs, 230
 - Device reset on exit, 214
 - Interrupt connections, 216
 - Locking a job in memory, 210
 - Mapping to physical addresses, 206
 - Physical address calculation, 212
 - Poke EMT, 203
 - Suspending/resuming execution, 212
- Rebooting the system, 85
- Record locking, 171
 - DIBOL, 248
 - See Shared files
- Redefining time-sharing lines, 3
- REENTER command, 86
- Reentrant run-times
 - See Shared run-time systems.
- Regions in extended memory, 167
- Releasing exclusive system control, 209
- REMOVE command, 86
- RENAME command, 47
- Requesting exclusive system control, 209
- RESET command, 47, 83, 86
- Resident run-times
 - See Shared run-time systems.
- RESORC, 247
- Restrictions
 - Keyboard commands, 86
 - Programs not supported, 247
- RESUME command, 86
- RMON
 - Real-time support consideration, 198
 - Simulated, 167
 - SYSGEN options word, 121
- RMON mapping, 45
- RT-11
 - Returning control to, 85
- Rubout filler character, 116

- RUN command, 47
 - /DEBUG switch, 45, 126
 - /HIGH switch, 45
 - /IOPAGE switch, 45
 - /LOCK switch, 46
 - /NONINTERACTIVE switch, 46, 138
 - /SINGLECHAR switch, 46, 119
- Run-time systems
 - See Shared run-time systems.
- Running programs, 44, 47
- SAVE command, 86
- Scheduling a completion routine, 229
- Scheduling of jobs, 2
- Scope type terminal support, 68
- SEND
 - DIBOL statement, 248
- SEND command, 48
- Sending messages, 140
- Serial devices
 - CL units, 3
- SET command, 48, 86
 - CACHE, 49
 - CCL, 49
 - CL units, 50
 - CORTIM, 54
 - CTRLD, 55
 - EDIT, 55
 - EMT, 55
 - ERROR, 56
 - Handler options, 48
 - HIPRCT, 56
 - IND, 57
 - INTIOC, 57
 - IO, 57
 - KMON, 24, 57
 - LANGUAGE, 59
 - LD, 59, 289
 - LOG, 59
 - LOGOFF, 60
 - MAXPRIORITY, 60
 - NUMDC, 61
 - PRIORITY, 61
 - PROMPT, 62
 - QUANxx, 62
 - SIGNAL, 63
 - SL, 63
 - TERMINAL, 65
 - TT, 65
 - TT ADM3A, 66
 - TT AUTOBAUD, 66
 - TT DEAD, 66
 - TT DECWRITER, 66
 - TT DEFER, 66
 - TT DIABLO, 67
 - TT ECHO, 67
 - TT EIGHTBIT, 67
 - TT FORM, 67
 - TT FORM0, 67
 - TT GAG, 67
 - TT HAZELTINE, 67
 - TT LA120, 67
 - TT LA36, 67
 - TT LC, 68
 - TT PHONE, 68
 - TT PRIVILEGE, 68
 - TT QUIET, 68, 89
 - TT QUME, 68
 - TT SCOPE, 68
 - TT SINGLE, 69
 - TT SPEED, 69
 - TT TAB, 69
 - TT TAPE, 69, 120
 - TT terminal-type, 123
 - TT VT100, 69
 - TT VT200, 70
 - TT VT50, 69
 - TT VT52, 69
 - TT WAIT, 70
 - UCI, 24
 - UCI=filnam, 24
 - UCL, 21, 70
 - UCL FIRST, 21, 70
 - UCL LAST, 21, 71
 - UCL MIDDLE, 21, 70
 - UCL NONE, 21, 71
 - VM, 71
 - WILDCARDS, 72
- SET UCL command
 - FIRST, 17
 - LAST, 18
 - MIDDLE, 17
 - NONE, 19
- SETSIZ program, 126, 251
- Setting processor priority level, 214
- Shared access to a file, 171
- Shared files, 171
 - Checking for modification of, 183

Index

- DIBOL, 248
- Opening, 171
- Protection modes, 172
- Saving channel status, 175
- Testing for locked blocks, 180
- Unlocking a block, 182
- Unlocking all locked blocks, 183
- Waiting for locked block, 178
- Shared run-time systems, 233
 - Associating with job, 233
 - Displaying run-times available, 79
 - Mapping into job region, 168, 235
- SHOW command, 72
 - ALL, 72
 - ALLOCATE, 73
 - ASSIGNS, 73
 - CACHE, 73
 - CL, 73
 - COMMANDS, 74
 - CONFIGURATION, 74
 - CORTIM, 75
 - DEVICES, 75
 - HIPRCT, 76
 - INTIOC, 76
 - JOBS, 76
 - MEMORY, 76
 - MOUNTS, 77
 - NUMDC, 77
 - PRIORITY, 78
 - QUANxx, 78
 - QUEUE, 78
 - RUN-TIMES, 79
 - SPOOL, 79
 - SUBSETS, 79, 289
 - TERMINALS, 79
 - USE, 81
- SHUTDOWN command, 85
- Signaling
 - System tuning parameters, 63
- Simulated RMON
 - Access through page 7, 167
 - Real-time support consideration, 198
- Simulated RMON mapping, 45
- Single character activation, 46, 69, 111, 119, 300
- Single line editor, 12
 - SET options, 63
- Skipping forward in a spool file, 104
- SL, 12
 - Handler, 247
 - SET command, 63
- Special Chain Exit, 24
- Speed
 - See terminal speed
- SPOOL command, 81, 102
 - Aligning a form, 103
 - Backing up in a spool file, 105
 - Checking device status, 105
 - Deleting queue entries, 104
 - HOLD & NOHOLD options, 106
 - SING & MULT options, 105
 - Skipping forward in a file, 104
 - Specifying a form name, 103
- Spooling, 101
 - Aligning a form, 103
 - Backing up in a file, 105
 - Checking device status, 105
 - Concept of, 101
 - Deleting queue entries, 104
 - Directing output to, 101
 - Displaying requests in queue, 78
 - Form names, 108
 - Holding output, 106
 - Number of free spool blocks, 159
 - Single and multifile processing, 105
 - Skipping forward in a file, 104
 - Specifying a form name, 103
 - Specifying default form name, 37
 - SPOOL command, 81
- Spooling, Operation of, 102
- SQUEEZE command, 82
- SRUN command, 86
- START command, 86
- Start-up command file, 9
- Starting detached jobs, 95
- STOP command, 85
- Stopping the system, 85
- Summaries of chapters, 4
- SUSPEND command, 86
- Symbolic instruction decoding, 293
- SYSGEN options word, 121
- SYSTAT command, 83

- System configuration
 - Showing, 74
- System device
 - Determining, 126
- System resource management, 2
- System tuning
 - SET SIGNAL, 63
- System values
 - Obtaining, 126
- Tab character support, 69
- Tape mode, 69, 120
- TECO, 55
 - MAKE command, 39
 - Use within command files, 90
- TECO command, 84
- Terminal control, 111
 - Break sentinel, 144
 - Character echoing, 117
 - Checking for activation, 147
 - Checking for input errors, 145
 - Command file input, 118
 - Defining activation characters, 117
 - Disabling virtual line use, 117
 - Echo control, 117
 - Field width activation, 118
 - Field width limit, 120
 - High-efficiency mode, 118, 157
 - Line-feed echoing, 120
 - Lower-case character input, 117
 - Non-wait TT input, 119
 - ODT activation mode, 159
 - Read time-out value, 143
 - Resetting activation characters, 118
 - Rubout filler character, 116
 - Single character activation, 119
 - Tape mode, 120
 - Transparency mode output, 118
 - VT52 & VT100 escape sequences, 116
- Terminal handler, 111
- Terminal logging, 59
- Terminal options
 - Setting, 150
- Terminal speed, 69
 - Setting, 151
- Terminal type
 - Determining, 123
- TIME command, 84
- Time-out value for TT reads, 143
- Time-sharing lines
 - Assigning to CL, 3, 152
- Transparency mode output, 118
- TSX-Plus
 - Determining if under, 121
- TSX-Plus license number
 - Determining, 126
- TSXPM program, 237, 238
- TSXUCL file
 - Size, 126
- TSXUCL program, 23
- Tuning parameters
 - Selecting, 63
- TYPE command, 84
- UCL, 20
- UCL command, 84
- UNLOAD command, 86
- Unlocking a job from memory, 211
- UNPROTECT command, 85
- Unrecognizable command, 18
- USE command, 85
- User Command Interface, 17, 24
- User Command Language, 20
- User name
 - Changing, 124
 - Determining, 124
 - Logging on, 9
- User-defined commands, 20
 - Order of interpretation, 17, 18, 19
- SHOW command, 74
- Utilities
 - Unsupported, 247
- Virtual arrays
 - FORTTRAN, 248
 - I/O page, 249
 - PAR 7 mapping, 230
- Virtual lines, 93
 - Comparison with detached jobs, 95
 - Disabling use of, 117
 - Execution priority of, 93
 - KED and K52, 94
 - Switching to, 11
- Virtual memory, 165
- Virtual programs, 167
 - Setting flag with SETSIZ, 253

Index

- Virtual region mapping, 206
- Virtual to physical address, 105, 212
- Virtual windows, 168
- VM
 - Handler, 169, 247
 - Initializing, 169
 - SET BASE command, 169
- VT100 support, 69, 116
- VT200 support, 65, 70, 116
- VT52 support, 69, 116
- VTCOM, 249
 - Using with CL units, 51, 53
- Watchpoint
 - Debugger, 293
- WHO command, 85
- X-off
 - See Control characters, Ctrl-S.
- X-on
 - See Control characters, Ctrl-Q.